

**Fermilab
Accelerator Division
Linac Department**

Beam Steering in the Fermilab Linac Upgrade

Fermilab Report - Linac Upgrade Note 211

Kevin L. Junck

May 20, 1993

I. Overview:

Beam steering in the Fermilab Linac has been working reliably for the past 10 years through the use of the L36 steering code written by Jim Smedinghoff. Based upon the measurements of two wire scanners, two dipole magnets are used to steer the beam to the proper position in the septum of the 200 MeV area for extraction to the Booster. Currently no consideration is given to beam position as it travels through the linac, the steering is done solely to ensure that the beam be in the correct position upon exiting the linac. In the side-coupled structure of the Linac Upgrade, the physical aperture through which the beam must pass is reduced from the current drift-tube radius of 2.2 cm to 1.5 cm. This 32% reduction means that the ability to correct beam position throughout the Linac Upgrade will be of importance. The console application program BEAST (BEAm STeering) has been written to utilize the increased number of Beam Position Monitors throughout the Linac Upgrade to achieve a global chi-square minimization of beam position errors. The code is written in the C programming language and a listing of the code is included in the appendix.

II. Theory:

A commonly used algorithm in the correction of closed orbits of circular machines is employed for the linac^{1,2}. In general we have N Beam Position Monitors and M dipole correcting magnets with $N > M$. The initial distorted path down the linac as measured by these N BPMs is

given by a vector \underline{X}_θ . Each element of this N-vector is simply the displacement from the desired position of the beam at each BPM. We wish to find an array of dimension M, denoted by $\underline{\theta}$, which consists of the angular kicks to be given by the dipole corrector magnets. The final particle position is then given by: $\underline{X}_f = \underline{X}_o + \underline{T} \cdot \underline{\theta}$ Where the NxM matrix T is : $T_{ij} = \frac{\partial x_i}{\partial \theta_j}$

namely, the change in beam position at BPM i caused by changing the strength of the angular kick at dipole j. We define the quantity χ^2 (chi-squared) by:

$$\chi^2 = |\underline{X}_f|^2 = (\underline{X}_o + \underline{T} \cdot \underline{\theta}) \cdot (\underline{X}_o + \underline{T} \cdot \underline{\theta}) = \sum_{i=1}^N \left(X_{oi} + \sum_{j=1}^M T_{ij} \theta_j \right)^2$$

To minimize χ^2 we set $\frac{\partial \chi^2}{\partial \theta_k} = 0$

$$\begin{aligned} \frac{\partial \chi^2}{\partial \theta_k} &= \sum_{i=1}^N 2 \left(X_{oi} + \sum_{j=1}^M T_{ij} \theta_j \right) T_{ik} \\ &= \sum_{i=1}^N 2 T_{ik} (\underline{X}_o + \underline{T} \cdot \underline{\theta})_i \\ &= 2 \sum_{i=1}^N \tilde{T}_{ki} (\underline{X}_o + \underline{T} \cdot \underline{\theta})_i \\ &= 2 \tilde{\underline{T}} \cdot (\underline{X}_o + \underline{T} \cdot \underline{\theta}) = 0 \end{aligned}$$

where the MxN matrix $\tilde{\underline{T}}$ is the transpose of matrix T.

Solving for the desired array $\underline{\theta}$ yields:

$$\underline{\theta} = -(\tilde{\underline{T}} \underline{T})^{-1} \tilde{\underline{T}} \underline{X}_o$$

Thus from measuring the matrix T and the beam position displacements \underline{X}_θ the proper deflections at the dipole corrector magnets $\underline{\theta}$ can be found.

III. Running the Code

Upon entering the page, there is a small delay while calibration data is read from disk. Upon completion of this, the user is presented with 4 options (Figure #1). Interrupting within the label of an option will activate that option. Any warnings or error messages will be displayed in the Messages box at the bottom of the screen. The 4 user options are:

1) "# Pulses to Avg" - allows the user to change the number of 15 Hertz beam pulses to be averaged together when reading the BPMs (Figure #2). The maximum number of pulses is 300 and is a parameter set within the code.

2) "* Read BPMs" - requests 15 Hz beam and reads the beam position from all active BPMs. A counter indicates the number of pulses that have been recorded. In order for a pulse to be considered "good", more than 30 mA of beam must be seen at the toroid leading to the momentum dump (L:TORMD). Upon completion of the BPM reading, the average beam position and standard deviation are shown (Figure #3). Current settings and readings from all active dipoles are also displayed.

3) "* Calculate New Dipole Settings" - uses the BPM data just measured to determine new settings for all of the active dipoles in order to correct the beam steering. The BPM reading display is removed and the offset (beam position error) at each BPM is shown. To eliminate noise from being introduced into the calculation, the offset is currently set to zero if the BPM reading is within 0.3 mm of the defined BPM zero. Also shown are the recommended changes in the dipole settings and the values of the new settings. If the beam is properly steered through the linac, all values of offset should be zero and thus the recommended change in dipole settings are also zero (Figure #4). Changes of more than 1 Amp in dipole current are shown in red, changes in the range of 0.1 Amps to 1 Amp are displayed in yellow while changes in dipole current less than 0.1 Amps are in green.

After the calculation is made, a new field "* Make New Settings *" appears. It is intended that interrupting within this field will automatically send the new settings and change the nominal value for the dipoles. This option is currently "work in progress" due to a conflict between the console applications routines of Brian Hendricks and the linac front end software of Bob Goodwin in regard to the manner in which alarm blocks are changed. Until this is resolved, new dipole settings and nominal values must be manually entered via a parameter page.

4) "*Check Calibration" - will begin a calibration run. As described in the theory section (see II above), the matrix T must be determined. This is simply measuring the change in position

of the beam at a BPM due to changing the current at a dipole. A calibration run consists of changing the dipole current 6 times in small steps about the present setting and recording the average BPM reading at all of the active BPMs. As the dipole setting is being changed, an informational message is printed in the messages box (Figure #5). A typical calibration run for the existing linac takes approximately 4 minutes when 30 beam pulses are averaged with 8 dipoles and 18 BPMs. At each point after BPM readings have been made, the standard BPM display (Figure #6) appears briefly on the screen. After this procedure is done for all of the active dipoles, the code then calculates the slope of the line of Average Beam Position versus Dipole Current Setting to determine the elements of the T matrix. Disabling of alarms, setting of dipoles, and the requesting of beam are all done automatically. A field "*** Abort Calibration**" appears directly under the "*** Check Calibration**" field to enable the user to abort this procedure. Upon an abort, all dipoles are returned to their initial settings and alarms are re-enabled.

After completion of the calibration procedure results are printed upon the screen (Figure #7). Depending upon the number of BPMs and dipoles, all of the information may not fit upon one screen. Interrupting in the "MORE" string underneath the "#Pulses to Avg" string will page through the calibration results. As the final page of calibration results is displayed, the string "MORE" changes to "DONE". The calibration results are compared to historical data and a warning message is displayed if the calibration has changed by more than 15%. Future calculations of dipole settings will use this new calibration data. To return to the "historical" calibration data the user should leave the page and re-enter.

Interrupting within the "Novice Level" string will change to "Expert Level" mode. Several more options are available at this level (Figure #8):

- 1) *** Change Active Elements** - presents a menu of all available BPMs and dipoles (Figure #9). Those elements that are displayed in green will be active in measurements and calculations while the elements displayed in yellow will not be considered by the code. Interrupting on the element name will change its status from active to inactive or vice versa. Interrupting on "DONE" ends the process.

2) "* Redefine BPM Zeros" - since the code tries to minimize the deviation from "normal" BPM readings, it must know what is "normal". Interrupting on a BPM name enables the user to enter in a new "zero" value for that BPM (Figure #10). These values can be saved in the file "RSX\$CONSOLE_APPLICAT_LINAC_MISC:PA1277BPM.D" by interrupting on the "SAVE" string.

3) "*Enter Calib Data" - allows the user to enter individual calibration data by hand. The data displayed is from the most recent calibration run, or the historical data if no calibration run has yet been done. Interrupting on any number allows the user to input a new number (Figure #11). After paging through all of the data (with the "MORE" string) the data may be saved in the file "RSX\$CONSOLE_APPLICAT_LINAC_MISC:PA1277CAL.D" by interrupting on the "SAVE" string. To maintain a common file structure, the data can be saved to file only if all BPMs and dipoles are active. Interrupting on "DONE" will return the user from editing mode with the calibration changes intact but not saved permanently to file.

IV. Algorithm Testing in the existing Linac

In order to test the effectiveness of this algorithm the beam was intentionally missteered by using the dipoles HT3IN and VT3IN. The code was then allowed to correct the beam position by using the dipoles HT4IN, HT5IN, HT7IN, and HT7OUT (along with the accompanying vertical dipoles). The region of active BPMs began at the exit of Tank 5 through the exit of Tank 9 as well as the BPMs in the 200 MeV area (BPH201, BPH202, BPV201, BPV202). Figures #12 - #16 show the results of these tests. The open circle represents the beam position error before the code's correction, and the closed box represents the position error after correction. In almost all cases, after only one or two iterations the code reduces the error to the noise level.

One curious feature involves the signal from BPH201. Average BPM readings from this BPM typically have a standard deviation that is 4-5 times larger than the others. Also from the calibration data of Figures #7a and #7d, changing horizontal dipole magnets has no effect upon the horizontal beam position measured at the BPM, while changing vertical dipole magnets has a large effect upon the horizontal beam position. Figure #17 shows the beam position as measured by the

wire scanners used in the L36 steering code before correction during the run Trial #5 (Figure #16). Figure #17 shows a larger vertical offset than horizontal offset while Figure #16 shows a larger horizontal offset. Based upon these two measurements, it is reasonable to conclude that BPH201 is actually measuring the vertical plane and BPV201 is measuring the horizontal plane. The large amount of noise in the measurement also indicates that there may be problems with the BPM pickup or amplifier.

V. Linac Upgrade Architecture and the Division of Responsibilities:

A general layout of the Linac Upgrade lattice is given in Table 1. The problem of beam steering can be divided into three regions:

- 1) Utilizing the transition section to correct beam position errors at the exit of the DTL before entrance into Module 1 of the Linac Upgrade. For this region, a specialized three-bump code may prove more effective in correcting the beam position.
- 2) Transport of beam through Module #1 to Module #6 of the Upgrade. The BEAST code described in this note has been designed for precisely this problem.
- 3) Steering of the beam for proper Booster extraction. This should also be within the abilities of the BEAST code by using as active elements the two dipole correctors of Module #7 and BPMs at the exit of Module #7 and in the 400 MeV line (analogous to the BPMs currently in the 200 MeV line on either side of the septum).

Console Location 1,
PA:29 JUNCK 1277

24-MAY-1993 11:25

29

Linac Steering

Novice Level

Pulses to Avg = 30 * Read BPMs
* Calculate New Dipole Settings * Check Calibration



Figure #1

Console Location 1,
PA:Z9 JUNCK 1277

24-MAY-1993 11:26

29 Linac Steering Novice Level

* Pulses to Avg = 30 * Read BPMs * Calculate New Dipole Settings * Check Calibration

of Beam Pulses to Average []

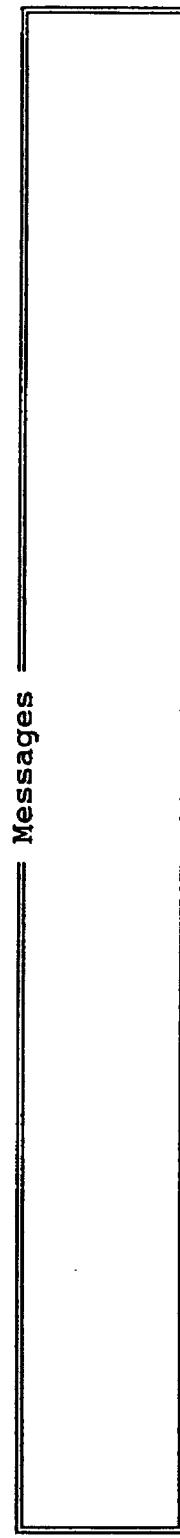


Figure H2

Console Location 10,
PA:z9 JUNCK 1277

24-MAY-1993 12:44

29

Linac Steering

Novice Level

| | | | | |
|---------------------------|---------------------------|----|---------------------------------|---------------------|
| # Pulses to Avg = 30 | * Read BPMs | 30 | * Calculate New Dipole Settings | * Check Calibration |
| BPH5OT = -4.15+/- 1.31 MM | BPV5OT = -7.29+/- .235 MM | | | |
| BPH6IN = .29 +/- 1.46 MM | BPV6IN = -6.88+/- .839 MM | | | |
| BPH6OT = 1.56+/- 1.98 MM | BPV6OT = -1.12+/- 1.78 MM | | | |
| BPH7IN = -.805+/- .804 MM | BPV7IN = -1.73+/- 1.48 MM | | | |
| BPH7OT = 3.8 +/- .6 MM | BPV7OT = 3.97+/- 2.36 MM | | | |
| BPH8IN = 2.98+/- 1.13 MM | BPV8IN = 4.06+/- 3.86 MM | | | |
| BPH8OT = -2.1 +/- 1.14 MA | BPV8OT = -1.52+/- .316 MM | | | |
| BPH9IN = -12 +/- .722 MM | BPV9IN = -3.21+/- .271 MM | | | |
| BPH9OT = 4.85+/- 1.76 MM | BPV9OT = -1.05+/- 1.39 MM | | | |
| BPH201 = 3.44+/- 5.84 MM | BPV201 = 6.71+/- .532 MM | | | |
| BPH202 = 14.9+/- 1.74 MM | BPV202 = 2.53+/- 1.17 MM | | | |

Messages

Figure #3

Console Location 10,
PA:Z9 JUNCK 1277

24-MAY-1993 12:44

29

Linac Steering

Novice Level

```
# Pulses to Avg = 30          * Read BPMs   30
                           * Calculate New Dipole Settings *
                           * Make New Settings *
OFFSET BPH5OT      0    BPV9IN   0 DIPOLE           CHANGE NEW SET
(mm)   BPH6IN      0    BPV9OT   0 SETTINGS HT3IN   0   0
          BPH6OT      0    BPV201   0           HT4IN   0   0
          BPH7IN      0    BPV202   0           HT5IN   0   .042
          BPH7OT      0           HT7IN   0   3.1
          BPH8IN      0           HT7OUT  0   -3.54
          BPH8OT      0           VT3IN   0   0
          BPH9IN      0           VT4IN   0   -2.14
          BPH9OT      0           VT5IN   0   0
          BPH201     0           VT7IN   0   -.5
          BPH202     0           VT7OUT  0   2.81
          BPV5OT      0
          BPV6IN      0
          BPV6OT      0
          BPV7IN      0
          BPV7OT      0
          BPV8IN      0
          BPV8OT      0
```

Messages =



Figure #4

Console Location 1,
PA:29 JUNCK

24-MAY-1993 11:39

29

Linac Steering

Novice Level

Pulses to Avg = 30

* Read BPMs 19

* Calculate New Dipole settings * Check Calibration *

| | | | |
|----------|----------|--------|-------|
| BPH6OT = | BPV6OT = | Set | Read |
| BPH7IN = | BPV7IN = | HT4IN | 2.4 |
| BPH7OT = | BPV7OT = | HT5IN | .042 |
| BPH8IN = | BPV8IN = | HT7IN | 3.1 |
| BPH8OT = | BPV8OT = | HT7OUT | -3.54 |
| BPH9IN = | BPV9IN = | VT4IN | -2.14 |
| BPH9OT = | BPV9OT = | VT5IN | 0 |
| BPH201 = | BPV201 = | VT7IN | -.5 |
| BPH202 = | BPV202 = | VT7OUT | 2.81 |
| | | | 2.69 |

Messages

Changing dipole setting for L:HT4IN to 2.400000
Changing dipole setting for L:HT4IN to 1.600000
Changing dipole setting for L:HT4IN to 0.800000
Changing dipole setting for L:HT4IN to 0.000000

Figure #5

Console Location 1,
PA:Z9 JUNCK 1277

24-MAY-1993 11:38

Z9

Linac Steering

Novice Level

| | | | | | |
|----------------------|-------------|----|---------------------------------|---------------------|----|
| # Pulses to Avg = 30 | * Read BPMs | 30 | * Calculate New Dipole Settings | * Check Calibration | |
| BPH6OT = -6.54+/- | 1.71 | MM | BPV6OT = -1.53+/- | 1.98 | MM |
| BPH7IN = -3.31+/- | .885 | MM | BPV7IN = -1.56+/- | 1.41 | MM |
| BPH7OT = -3.67+/- | .591 | MM | BPV7OT = 4.38+/- | 2.27 | MM |
| BPH8IN = -9.27+/- | 1.11 | MM | BPV8IN = 4.9 +/- | 5.32 | MM |
| BPH8OT = -.775+/- | 1.05 | MA | BPV8OT = -.522+/- | .243 | MM |
| BPH9IN = -13.9+/- | .712 | MM | BPV9IN = -2.34+/- | .451 | MM |
| BPH9OT = 6.62+/- | 1.55 | MM | BPV9OT = .422+/- | 1.83 | MM |
| BPH201 = 7.16+/- | 6.5 | MM | BPV201 = .344+/- | .578 | MM |
| BPH202 = 3.94+/- | 2.02 | MM | BPV202 = 1.09+/- | 1.26 | MM |

Changing dipole setting for L:HT4IN to -1.600000
Changing dipole setting for L:HT4IN to -2.400000
Everything is fine
Okay, but be careful.

Messages

Figure #6

Console Location 1,
PA:Z9 JUNCK 1277

24-MAY-1993 11:46

| | Linac Steering | Novice Level |
|--------|---|---|
| Z9 | | |
| MORE | # Pulses to Avg = 30 | * Read BPMs 30 |
| HT3IN | BPH5OT BPH6IN BPH6OT BPH7IN BPH7OT BPH8IN BPH8OT BPH9IN BPH9OT BPH201 -1.81 2.19 1.07 .764 -.53 -.585 -1.28 -1.49 2.49 0 | * Calculate New Dipole Settings * Check Calibration |
| HT4IN | 0 -1.22 3.29 1.04 2.87 5.23 -.512 .803 -.816 0 | |
| HT5IN | 1.27 -.875 -3.46 -1.38 -1.81 -3.56 1.43 .527 -1.39 0 | |
| HT7IN | 0 0 0 0 1.85 2.95 1.01 1.6 -2.64 0 | |
| HT7OUT | 0 0 0 0 0 0 2.46 2.42 -4.36 0 | |
| VT3IN | 0 0 0 0 0 0 0 0 .624 5.39 | |
| VT4IN | 0 0 0 0 0 0 .314 0 0 0 | 4.1 |

Changing dipole setting for L:VT7OUT to 5.212805
Changing dipole setting for L:VT7OUT to 4.412805
Changing dipole setting for L:VT7OUT to 3.612805
Changing dipole setting for L:VT7OUT to 2.812805

Figure #7a

Console Location 1,
PA:29 JUNCK 1277

24-MAY-1993 11:46

29

Linac Steering

Novice Level

```
# Pulses to Avg = 30          * Read BPMs      30
MORE                      * Calculate New Dipole Settings * Check Calibration
BPH202 BPV50T BPV6IN BPV60T BPV7IN BPV70T BPV8IN BPV80T BPV9IN BPV90T
HT3IN    1.18   0     0     0     0     0     0     0     0     0     0
HT4IN    4.57   0     .265   0     0     0     0     0     -.326   -.347   -.752
HT5IN    -4.43   0     0     0     0     0     0     0     0     0     0
HT7IN    1.29   0     0     0     0     0     0     0     0     0     0
HT7OUT   -3.35   0     0     0     0     0     0     0     0     0     0
VT3IN    0     0     -.735   2.4   1.33   1.89   4.6   -.775   0     1.6
VT4IN    0     -1.81  -2.06  -.521  -.754   0     0     2.48   2.45   2.3
```

Changing dipole setting for L:VT7OUT to 5.212805
Changing dipole setting for L:VT7OUT to 4.412805
Changing dipole setting for L:VT7OUT to 3.612805
Changing dipole setting for L:VT7OUT to 2.812805

Figure #7b

Console Location 1,
PA:29 JUNCK 1277

24-MAY-1993 11:47

29 Linac Steering Novice Level

```
# Pulses to Avg = 30          * Read BPMs 30
MORE          * Calculate New Dipole Settings * Check Calibration
BPV201 BPV202
HT3IN        -1.18   0
HT4IN        2.63   .489
HT5IN        -1.11   0
HT7IN        2.1    0
HT7OUT       1.44   0
VT3IN        -.357  -.883
VT4IN        0      -1.7
```

Changing dipole setting for L:VT7OUT to 5.212805
Changing dipole setting for L:VT7OUT to 4.412805
Changing dipole setting for L:VT7OUT to 3.612805
Changing dipole setting for L:VT7OUT to 2.812805

Figure #7c

Console Location 1,
PA:Z9 JUNCK 1277

24-MAY-1993 11:47

29

Linac Steering

Novice Level

```
# Pulses to Avg = 30          * Read BPMs   30
MORE      * Calculate New Dipole Settings * Check Calibration
          BPH5OT BPH6IN BPH6OT BPH7IN BPH7OT BPH8IN BPH8OT BPH9IN BPH9OT BPH201
VT5IN     0      0      0      0      0      0      0      0      0      0      -.627    -7.98
          VT7IN     0      0      0      0      0      0      0      0      0      -.513    -6.01
          VT7OUT    0      0      0      0      0      0      0      0      0      0      -4.35
```

Changing dipole setting for L:VT7OUT to 5.212805
Changing dipole setting for L:VT7OUT to 4.412805
Changing dipole setting for L:VT7OUT to 3.612805
Changing dipole setting for L:VT7OUT to 2.812805

Figure 47d

Console Location 1,
PA:Z9 JUNCK 1277

24-MAY-1993 11:47

| | | | |
|--------|---|---|--------------|
| | | Linac Steering | Novice Level |
| | | <hr/> | |
| | | 29 | |
| | | <hr/> | |
| | | # Pulses to Avg = 30 | |
| | | * Read BPMs ³⁰ | |
| MORE | | * Calculate New Dipole Settings * Check Calibration | |
| | | BPH202 BPV50T BPV6IN BPV60T BPV7IN BPV70T BPV8IN BPV80T BPV9IN BPV90T | |
| VT5IN | 0 | 1.87 | -2.24 |
| | | -.896 | -2.47 |
| VT7IN | 0 | 0 | 0 |
| | | 0 | 0 |
| VT7OUT | 0 | 0 | 0 |
| | | 0 | 0 |

| | |
|--|----------|
| Changing dipole setting for L:VT7OUT to 5.212805 | Messages |
| Changing dipole setting for L:VT7OUT to 4.412805 | |
| Changing dipole setting for L:VT7OUT to 3.612805 | |
| Changing dipole setting for L:VT7OUT to 2.812805 | |

Figure #7e

Console Location 1,
PA:Z9 JUNCK 1277

24-MAY-1993 11:48

Z9

Linac Steering

Novice Level

```
# Pulses to Avg = 30          * Read BPMs 30
DONE                                * Calculate New Dipole Settings * Check Calibration
                                         BPV201 BPV202
VT5IN      .389   1.87
                                         BPV201 BPV202
VT7IN      .291   1.83
                                         BPV201 BPV202
VT7OUT     0       2.3
```

Changing dipole setting for L:VT7OUT to 5.212805
Changing dipole setting for L:VT7OUT to 4.412805
Changing dipole setting for L:VT7OUT to 3.612805
Changing dipole setting for L:VT7OUT to 2.812805

Messages

Console Location 1,
PA:29 JUNCK 1277

24-MAY-1993 11:31

| Linac Steering | Expert Level |
|--|---|
| * Change Active Elements # Pulses to Avg = 30 | * Read BPMs * Calculate New Dipole Settings * Check Calibration |
| | * Enter Calib Data * Redefine BPM Zeros |

z9

Messages

Okay, but be careful.

Figure #8

Z9

Linac Steering

Expert Level

* Change Active Elements * Read BPMs 300
Pulses to Avg = 300 * Calculate New Dipole Settings * Enter Calib Data
* Redefine BPM Zeros * Check Calibration

Green = Active Element
DONE

| | | |
|----------|----------|----------|
| L:BPH5OT | L:BPV5OT | L:HT3IN |
| L:BPH6IN | L:BPV6IN | L:HT4IN |
| L:BPH6OT | L:BPV6OT | L:HT5IN |
| L:BPH7IN | L:BPV7IN | L:HT7IN |
| L:BPH7OT | L:BPV7OT | L:HT7OUT |
| L:BPH8IN | L:BPV8IN | L:VT3IN |
| L:BPH8OT | L:BPV8OT | L:VT4IN |
| L:BPH9IN | L:BPV9IN | L:VT5IN |
| L:BPH9OT | L:BPV9OT | L:VT7IN |
| L:BPH201 | L:BPV201 | L:VT7OUT |
| L:BPH202 | L:BPV202 | |

Okay, but be careful.

Messages

Figure #9

Console Location 1,
PA: Z9 JUNCK 1277

24-MAY-1993 11:29

Z9

Linac Steering

Expert Level

* Change Active Elements * Read BPMs 300
Pulses to Avg = 300 * Calculate New Dipole Settings * Check Calibration

| DONE | SAVE |
|--------|-------|
| BPH5OT | -1.47 |
| BPH6IN | -2.24 |
| BPH6OT | -3.73 |
| BPH7IN | -3.2 |
| BPH7OT | 4.32 |
| BPH8IN | 3.44 |
| BPH8OT | -2.04 |
| BPH9IN | -11.8 |
| BPH9OT | 4.5 |
| BPH201 | 4.13 |
| BPH202 | 15.3 |

Enter New Value For BPM []

Okay, but be careful.

Messages

Figure #10

Console Location 1,
PA:29 JUNCK

24-MAY-1993 11:49

Z9

Linac Steering

Expert Level

| | Change Active Elements | Read BPMs | 30 | * Enter Calib Data | | | | | | | |
|--------|------------------------|-----------|---------------------------------|----------------------|--------|--------|--------|--------|--------|--------|---|
| | # Pulses to Avg = | 30 | * Calculate New Dipole Settings | * Redefine BPM Zeros | | | | | | | |
| MORE | | | | * Check Calibration | | | | | | | |
| HT3IN | BPH5OT | BPH6IN | BPH6OT | BPH7IN | BPH7OT | BPH8IN | BPH8OT | BPH9IN | BPH9OT | BPH201 | |
| | -1.81 | 2.19 | 1.07 | .764 | -.53 | -.585 | -.1.28 | -.1.49 | 2.49 | 0 | |
| HT4IN | 0 | -1.22 | 3.29 | 1.04 | 2.87 | 5.23 | -.512 | .803 | -.816 | 0 | |
| HT5IN | 1.27 | -.875 | -3.46 | -1.38 | -1.81 | -3.56 | 1.43 | .527 | -1.39 | 0 | |
| HT7IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1.6 | -2.64 | 0 | |
| HT7OUT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 46 | 2.42 | -4.36 | 0 |
| VT3IN | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | .624 | 5.39 | |
| VT4IN | 0 | 0 | 0 | 0 | 0 | .314 | 0 | 0 | 0 | 4.1 | |

Messages

Okay, but be careful.
calibration discrepancy, using newer data
calibration discrepancy, using newer data
calibration discrepancy, using newer data

figure #11

Trial #1 : Ht3in = -2 A, Vt3in = 1A

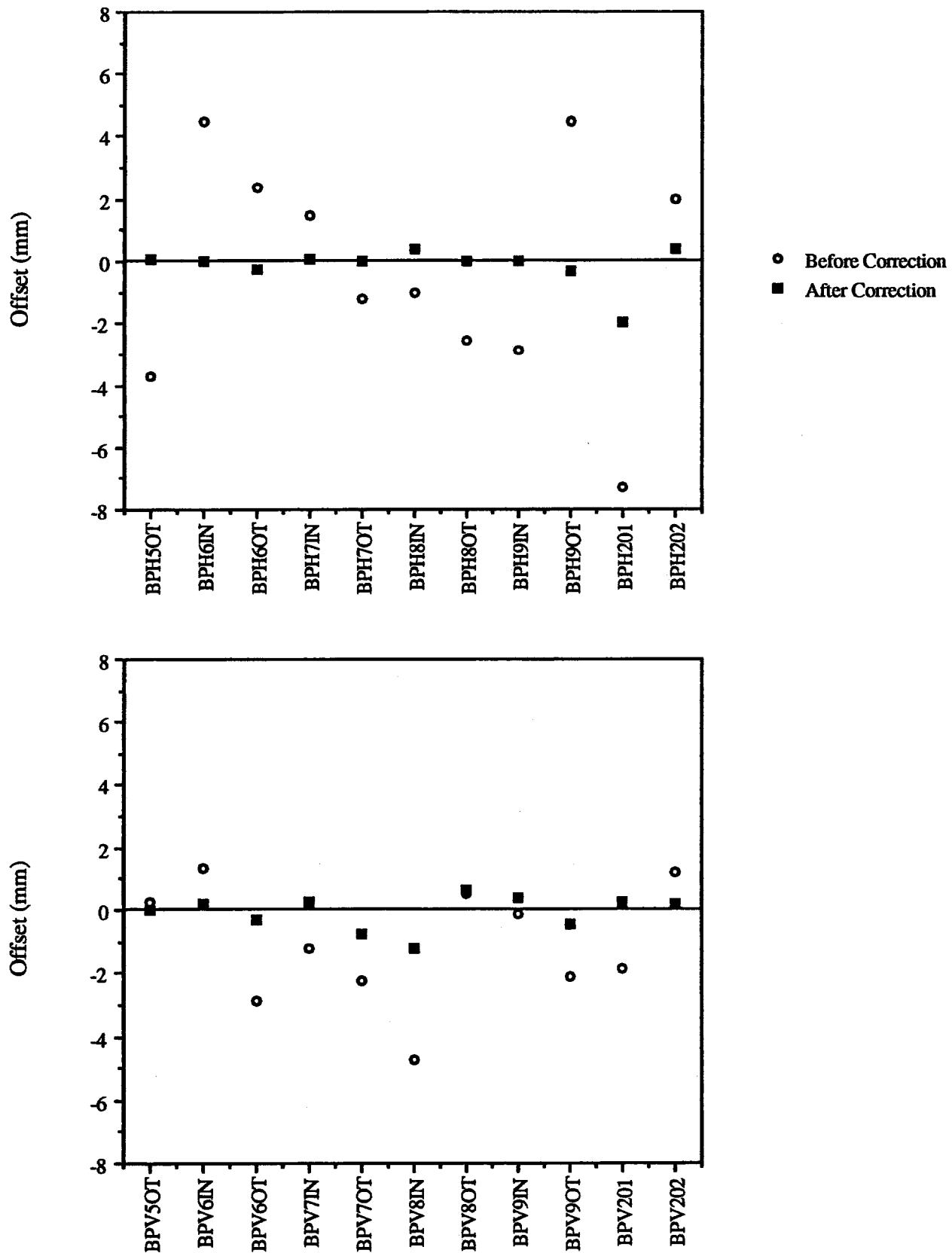


Figure #12

Trial #2 : Ht3in = 2 A; Vt3in = 3 A

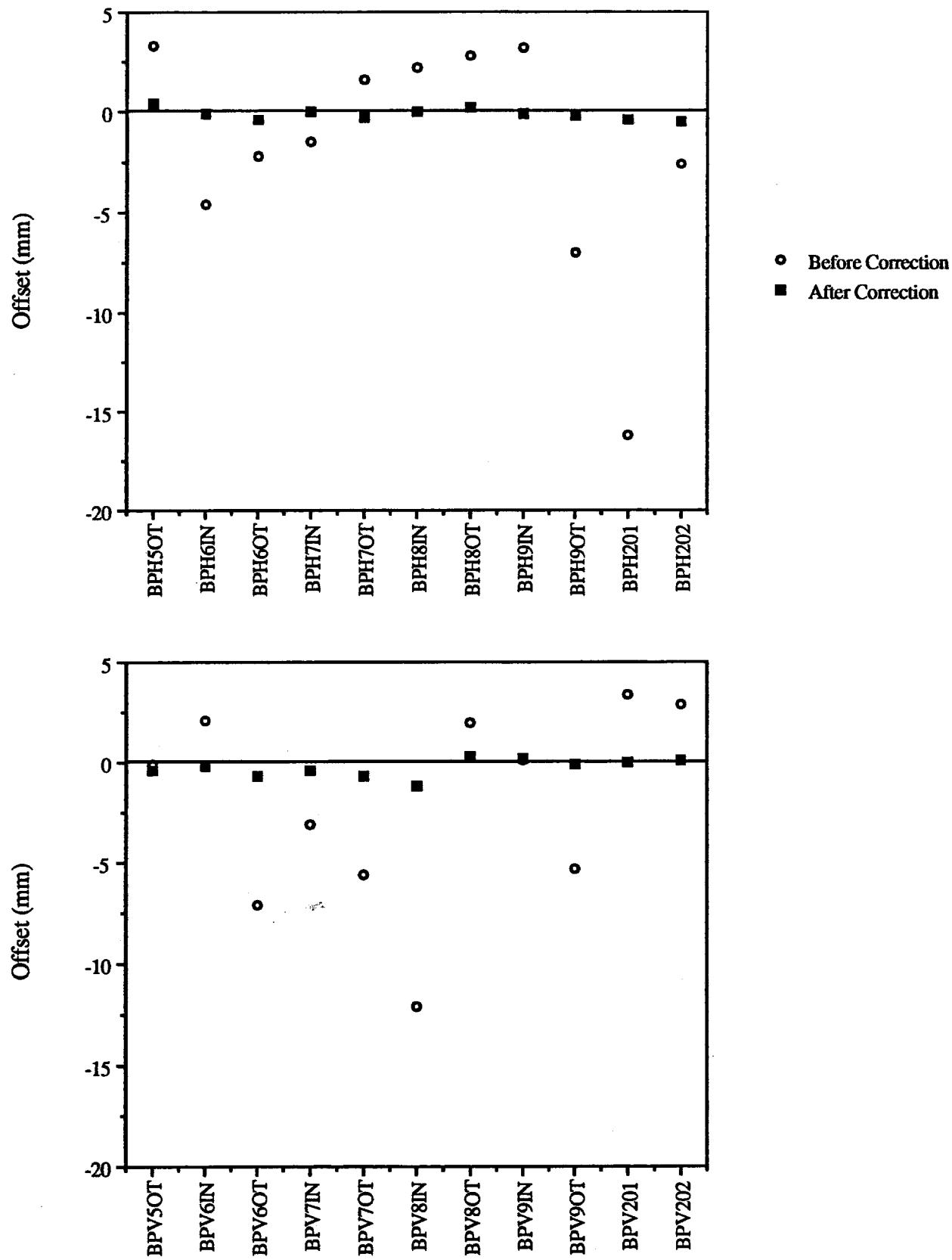


Figure #13

Trial #3 : Ht3in = -2 A; Vt3in = -2 A

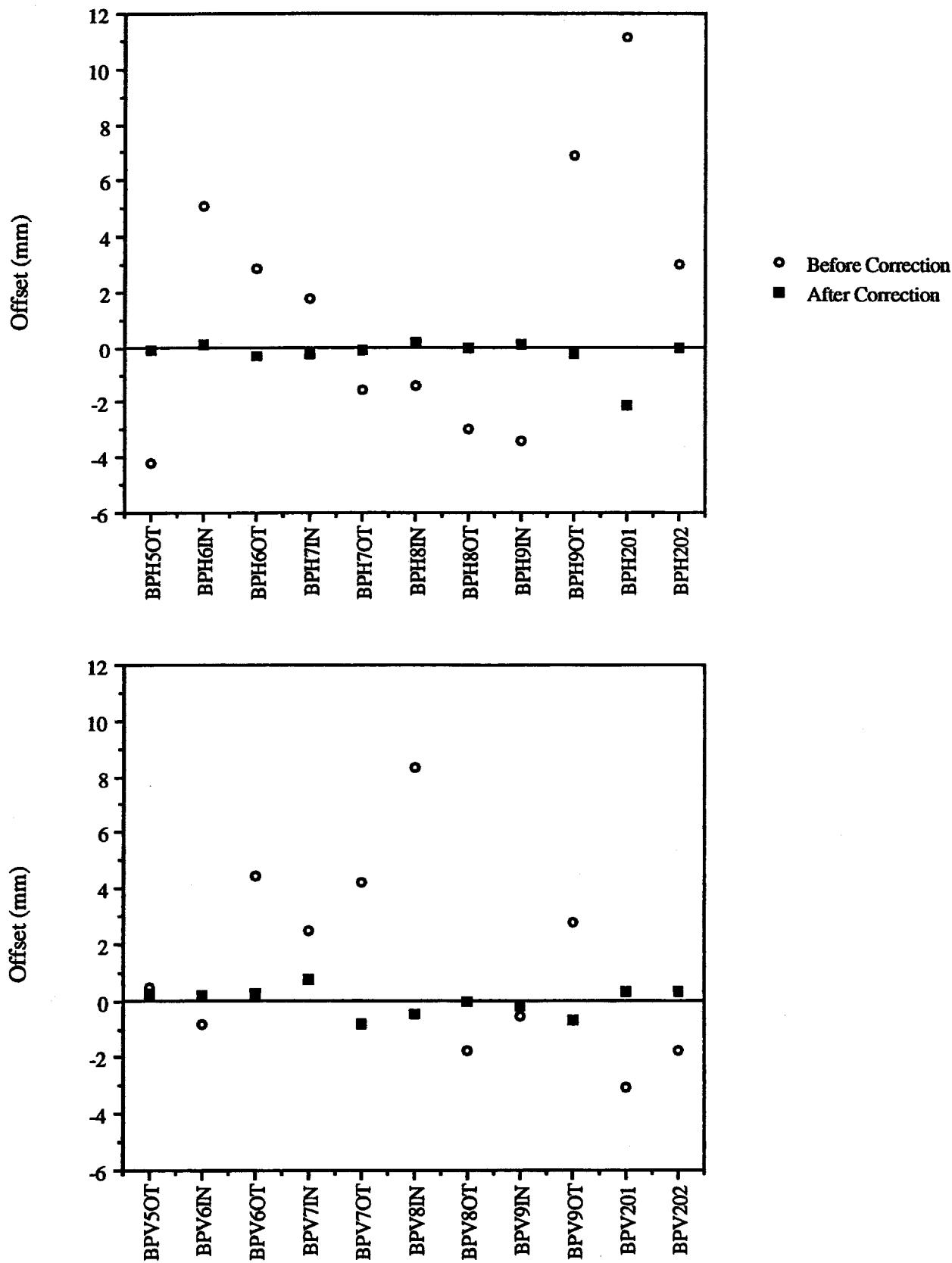


Figure #14

Trial #4 : Ht3in = -1 A; Vt3in = 1 A

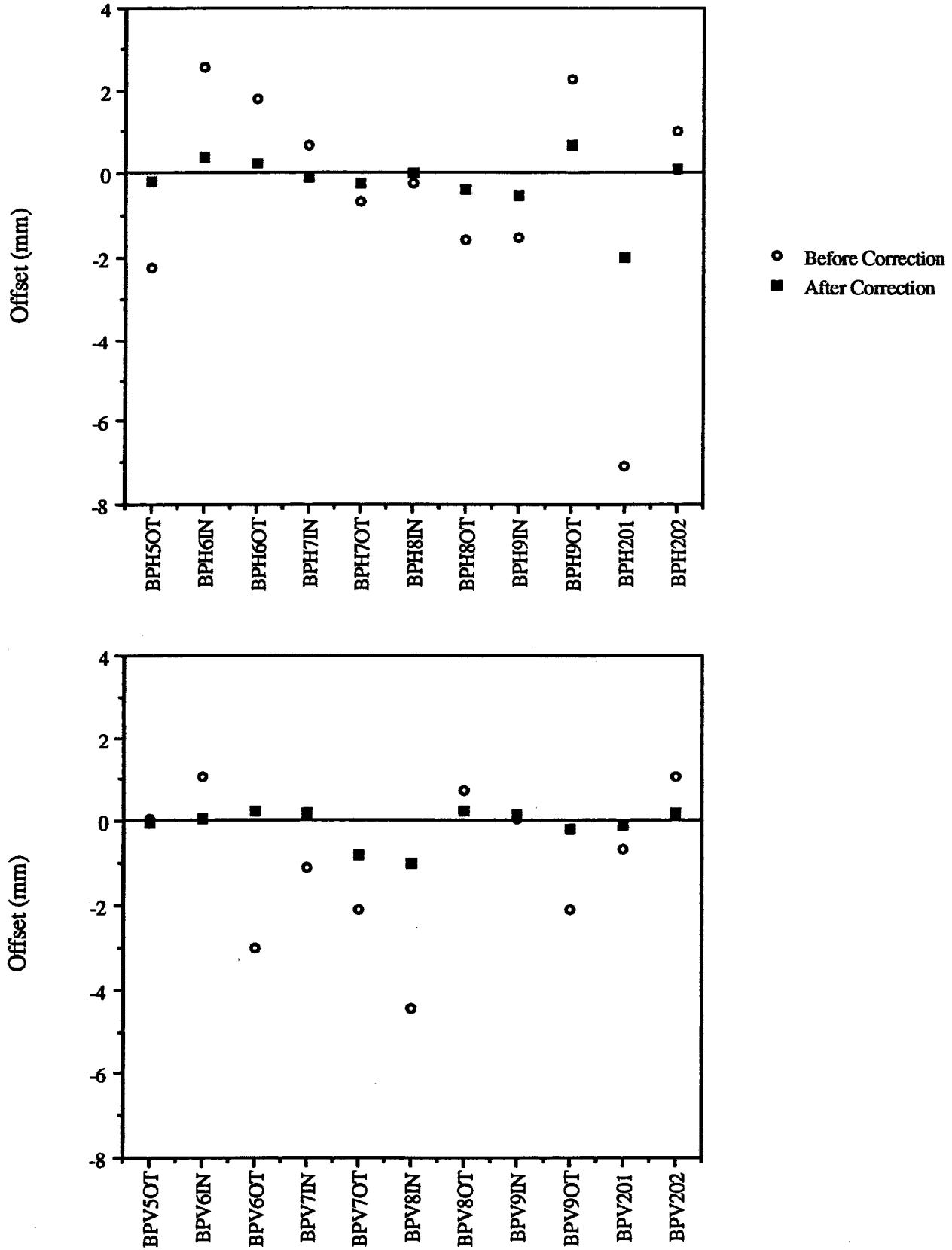


Figure #15

Trial #5 : Ht3in = -2 A; Vt3in = 2 A

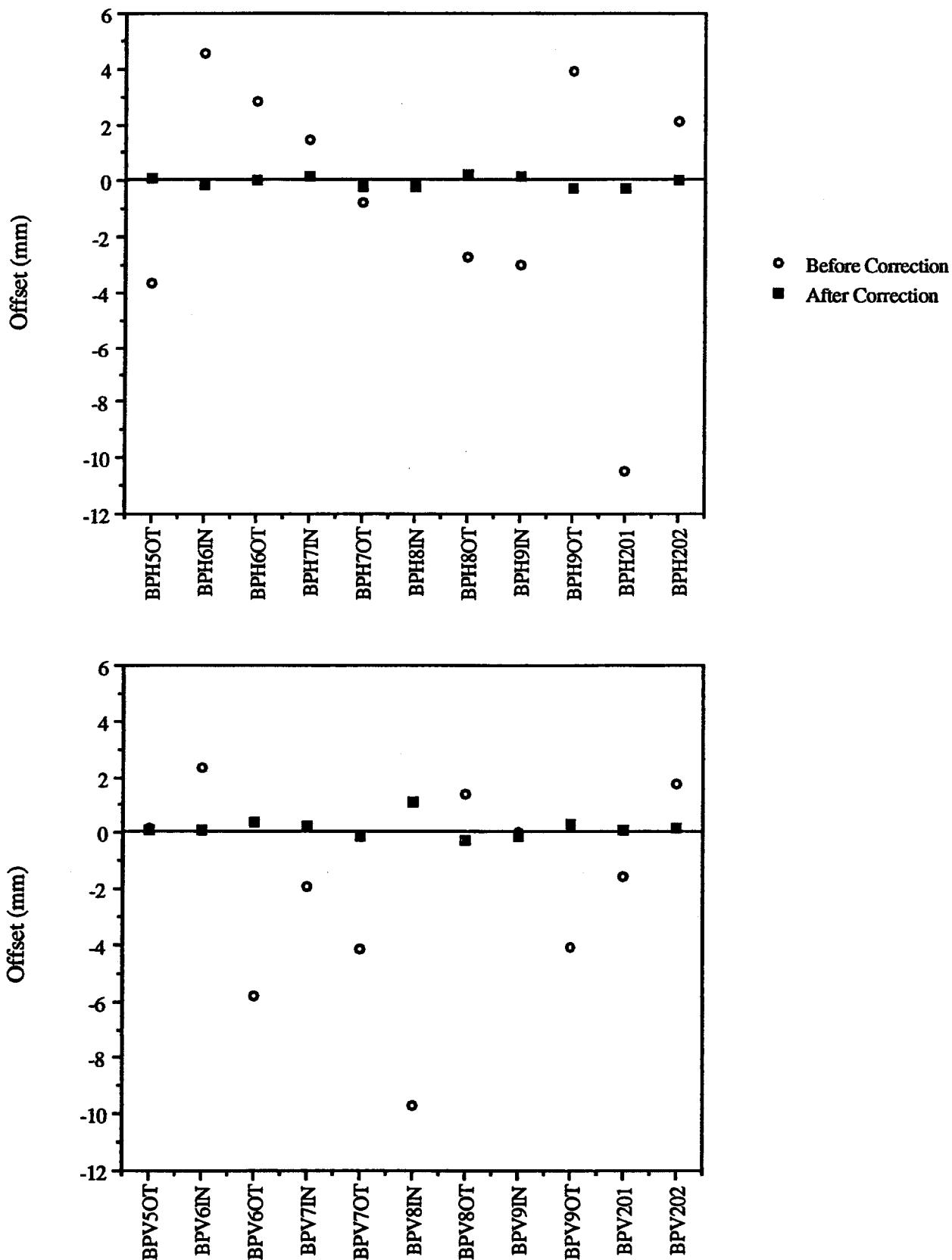
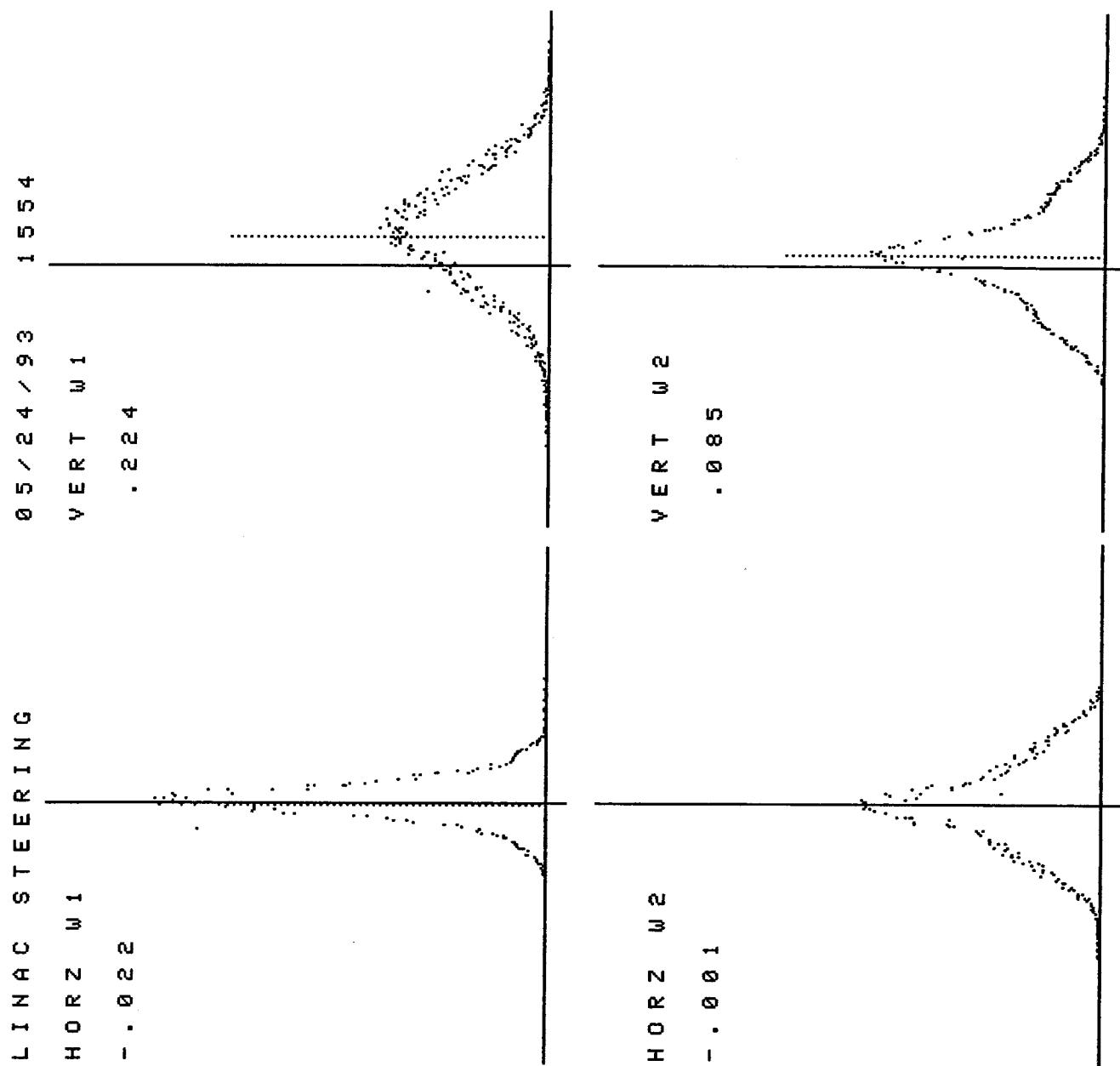


Figure #16

Console Location 1,
Storage Scope Emulation

24-MAY-1993 15:55



Trial #5 (see Figure #1)

Figure #17 (before)
correction

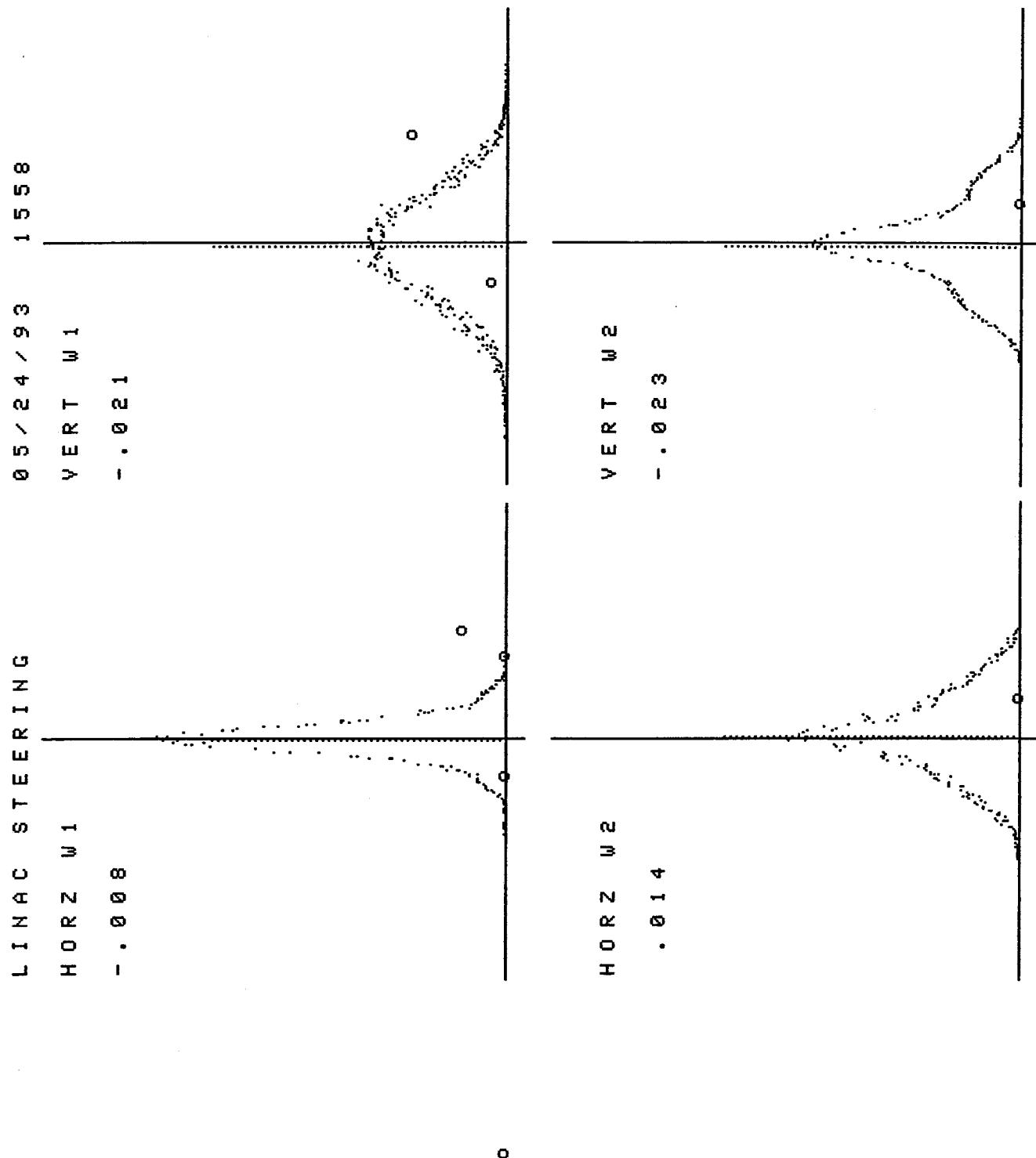


Figure #18 (after)
correction

Table #1 - Linac Upgrade Lattice - Focussing and Steering Elements

Existing DTL Linac:

Dipole Corrector between Tank 4 and Tank 5 (HT5IN, VT5IN)
BPM (BPH5IN, BPV5IN; BPH5OT, BPV5OT)

Transition Section:

Dipole Corrector
H Quad with BPM
RF Buncher
V Quad with BPM
Wire Scanner
Dipole Corrector
RF Buncher
H Quad with BPM
Wire Scanner
Dipole Corrector

Module #1 - #7

Dipole Corrector - Module #1 only
V Quad - with BPM Module #1, #4-#7; no BPM Modules #2 & #3
RF Tank 1
H Quad with BPM
Dipole Corrector
RF Tank 2
V Quad with BPM
Wire Scanner
RF Tank 3
H Quad with BPM
Wire Scanner
RF Tank 4

References:

¹R. Talman, "A Universal Algorithm for Accelerator Correction", AIP Conference Proceedings 255, A. Chao ed., AIP, NY 1992, p. 66.

²A.S. King, et al, "Closed Orbit Correction in SPEAR", IEEE Transactions on Nuclear Science, Vol NS-20 1973, p. 898.

Appendix I:

Listing of the code PA1277.C follows.

```
/*
** LINAC UPGRADE BEAST (BEAm Steering)
**
** Kevin L. Junck 3/3/93 - 5/12/93
** AD/Linac x8368
**
** ABSTRACT:
** Using BPMs and Dipole Magnets to steer linac beam
** Dipole Steering Magnets : HT4IN HT5IN HT7IN HT7OUT (&VT...)
** BPMs : BPH5OT BPH6IN BPH6OT BPH7IN BPH7OT BPH8IN BPH8OT BPH9IN
** BPH9OT BPH201 BPH202 (and corresponding BPV...)
*/
*****
```

```
#include "cnsparam.h"           /* generic console constants */
#include "cns_data_structs.h"    /* generic console data structures */
#include "dbprops.h"             /* database properties */
#include "clib.h"                /* 'old' CLIB prototypes */
#include "cbslib.h"              /* CBS library defs */
#include "diolib.h"              /* DIO library defs */
#include "acnet_errors.h"         /* defined constants for ACNET errors */
#include "tclk_events.h"          /* defined constants for TCLK events */
#include "ul$cbsaux:auxlib.h"     /* AUX library defs */
#include <math.h>                /* for sqrt */

/* define some important numbers */
#define maxbpmp      22 /* max number of BPMs */
#define maxdipolep   10 /* max number of dipoles */
#define maxavgp     300 /* max number of readings to average together */
#define numcalibp    6  /* number of BPM/dipole calib points */
#define beamwaitp   40000 /* counter limit to abort bpm reading */

/* define location of some fields on the window */
#define nrowsp       30      /* # of rows in background window */
#define ncoulsp      80      /* # of columns in bkgd window */

#define nerrrow      4       /* # of rows in error message window */
#define ROW_ERR      nrowp-nerrrow-1 /* error message window parameters */
#define COL_ERR      1
#define LEN_ERR      ncoulsp-2

#define liststr      " * Change Active Elements"
#define row_list     4
#define col_list     2

#define avgstr       "# Pulses to Avg = "
#define row_avg      5
#define col_avg      2

#define checkstr     " * Read BPMs"
#define row_check    4
#define col_check    27

#define calcstr      " * Calculate New Dipole Settings"
#define makestr      " * Make New Settings"
#define row_calc     5
#define col_calc     27

#define zerostr      " * Redefine BPM Zeros"
#define row_zero     4
#define col_zero     59

#define cdatstr      " * Enter Calib Data"
#define row_cdat     3
```

93/05/24
13:22:10

pal277.c

2

```
#define col_cdat      59
#define calibstr        "Check Calibration"
#define row_calib       5
#define col_calib       59

#define row_disp        7
#define col_disp        2
#define len_disp        5 /* # of characters to display per entry */
#define moveover        3

#define titlestr        "Linac Steering"
#define novicestr       "Novice Level"
#define expertstr       "Expert Level"

/* global data */
static short    int_wid;           /* interrupt window ID */
static short    home_wid;          /* home window ID */
static short    int_type;          /* interrupt type */
static int      int_row, int_col;  /* interrupt location data */
static int      int_info;          /* special interrupt information */
static short    piread = -PRREAD;   /* device property (reading) */
static short    piset = -PRSET;     /* device property (setting) */
static short    ftd = FTD_15HZ;    /* 15 Hz read */
static int      torind = 0;        /* device index for toroid */
static int      numavg = 30;       /* initialize number of averages */
static float    tor;              /* toroid reading to see if there was beam */
static float    curlimit = 5.90;   /* limit for dipole current */
/* average and standard deviation of bpm readings, index is bpm number */
static double   avgbpm[maxbpm], stddev[maxbpm];
/* bpmzero is data from disk, zdata is condensed to num bpms in use */
static float    bpmzero[maxbpm], zdata[maxbpm];
static double   curstep = 0.80;    /* step size in Amps for dipole calibration */
/* slope is live calibration data, hist is historical data */
/* cdata is what get's used by the calculation section */
static float    slope[maxbpm+1][maxdipole+1], sigslope[maxbpm+1][maxdipole+1];
static float    cdata[maxbpm+1][maxdipole+1], hist[maxbpm+1][maxdipole+1];
static int      beam_enable_di=0; /* device index for enabling beam */
static int      numdipole = maxdipole; /* active number of dipoles */
static int      numbpm = maxbpm; /* active number of bpms */
/* flag to determine if calibration has been done */
static int      calibflag=0;
/* flag to determine if bpm readings have been taken */
static int      rbpmflag=0;
/* flag to go to expert level */
static int      expert=0;
/* flag to make sure we're ready to make new dipole settings */
static int      makeflag=0;
/* signal to makelist() that it is only initialization run */
static int      firsttime=0;

static char    bpmnames[maxbpm][DEVNAM_LEN+1]; /* array of all bpm names */
static char    bpms[DEVNAM_LEN*maxbpm+1]; /* bpm active device names */
static int     bdindex[maxbpm]; /* bpm device index array */
static short   berrors[maxbpm]; /* errors array */
static float   bpmvalues[maxbpm]; /* values of devices */
static char    bunits[maxbpm*LEN_DEV_UNITS+1]; /* device units */
static int     bpmlist; /* list id number */

static char    dipnames[maxdipole][DEVNAM_LEN+1]; /* array of all dip names */
static char    dipoles[DEVNAM_LEN*maxdipole+1]; /* active dipole device names */
static int     ddindex[maxdipole]; /* dipole device index array */
static short   derrors[maxdipole]; /* errors array */
static float   dipvalues[maxdipole]; /* values of devices */
```

93/05/24
13:22:10

3

pa1277.c

```
static char      dunits[maxdipole*LEN_DEV_UNITS+1]; /* device units */
static int       diplist;    /* list id <number for reading dipoles */
static int       dipsetlist; /* list id number for setting dipoles */
static int       dsetrlist;  /* list id number for reading the dipole settings */

static int       bpmflag[maxbpm]; /* array of active bpms: 1=active,0=not */
static int       dipflag[maxdipole]; /* array of active dipoles */

/* global routines */
static void      pgm_ini(void);      /* initialization interrupt handler */
static void      pgm_trm(void);      /* termination interrupt handler */
static void      pgm_per(void);      /* periodic interrupt handler */
static void      pgm_kbd(void);      /* keyboard interrupt handler */
static int       readbpm(void);      /* request beam and read bpms */
static void      makelist(void);     /* make up device index lists */
static void      readcalib(void);    /* read calibration data from disk */
static void      readzero(void);     /* read bpm zeros from disk */
static void      makezero(void);     /* change bpm zeros */
static void      makecalib(void);    /* change calib data by hand */
static void      displaycalib();    /* display calib data on screen */

/*****************************************/
/* main - Get interrupt type and call routine to decode it */

main()
{
    while (TRUE) { /* do forever */
        window_intype(&int_wid,&int_type,
                      &int_row,&int_col,&int_info); /* get interrupt info */
        switch (int_type) {
            case INTINI:          /* initialization interrupt */
                pgm_ini();
                break;
            case INTTRM:          /* termination interrupt */
                pgm_trm();
                break;
            case INTKBD:           /* keyboard interrupt */
                pgm_kbd();
                break;
            case INTPER:           /* periodic interrupt */
                pgm_per();
                break;
            default:
                break;
        } /* switch */
    } /* while */
} /* main */

/*****************************************/
/* pgm_ini - Initialize program */

static void pgm_ini(void)
{
    char tempstr[40];
    int i,status;

    /* save window id in home_wid */
    home_wid = int_wid;

    /* set window size */
    window_set_background_size_c(nrows,ncols);

    /*setup error report*/
    error_init_c(ROW_ERR,COL_ERR,LEN_ERR,LOG_LCL,0,0,nerrrow);

    dio_tuner(); /* initialize DIO and turn on logging */
}
```

93/05/24
692810

pal277.c

4

```
fm_tuner(); /* initialize file manager routines and turn on logging */

/* put up title */
window_center_text_c(WMNGR_BACKGROUND,1,titlestr,strlen(titlestr),CYAN);

/* display novice commands background window */
btvm_c(row_avg,col_avg,avgstr,strlen(avgstr),GREEN);
sprintf(tempstr,"%d",numavg);
btvm_c(row_avg,col_avg+strlen(avgstr),tempstr,strlen(tempstr),GREEN);
btvm_c(row_check,col_check,checkstr,strlen(checkstr),GREEN);
btvm_c(row_calc,col_calc,calcstr,strlen(calcstr),GREEN);
btvm_c(row_calib,col_calib,calibstr,strlen(calibstr),GREEN);
btvm_c(1,ncols-strlen(novicestr)-2,novicestr,strlen(novicestr),GREEN);

/* setup beam enable bit */
status = dio_device_index("L:BEAMEN",&beam_enable_di);
if (status != DIO_SUCCESS)
    error_message_c("Error retrieving beam enable index",0,RED,TRUE);

/* get device index for Toroid */
status = dio_device_index("L:TORMD ",&torind);
if (status != DIO_SUCCESS) { /* failed in retrieving device index */
    error_message_c("Error retrieving device index TO6IN",0,RED,TRUE); }

/* initialize flag lists to include all possible bpms and dipoles */
for (i=0;i<=maxbpm-1;i++) {
    bpmflag[i]=1;
    dipflag[i]=1; }

/* store names of all bpms */
strcpy(bpmnames[0],"L:BPH5OT");
strcpy(bpmnames[1],"L:BPH6IN");
strcpy(bpmnames[2],"L:BPH6OT");
strcpy(bpmnames[3],"L:BPH7IN");
strcpy(bpmnames[4],"L:BPH7OT");
strcpy(bpmnames[5],"L:BPH8IN");
strcpy(bpmnames[6],"L:BPH8OT");
strcpy(bpmnames[7],"L:BPH9IN");
strcpy(bpmnames[8],"L:BPH9OT");
strcpy(bpmnames[9],"L:BPH201");
strcpy(bpmnames[10],"L:BPH202");
strcpy(bpmnames[11],"L:BPV5OT");
strcpy(bpmnames[12],"L:BPV6IN");
strcpy(bpmnames[13],"L:BPV6OT");
strcpy(bpmnames[14],"L:BPV7IN");
strcpy(bpmnames[15],"L:BPV7OT");
strcpy(bpmnames[16],"L:BPV8IN");
strcpy(bpmnames[17],"L:BPV8OT");
strcpy(bpmnames[18],"L:BPV9IN");
strcpy(bpmnames[19],"L:BPV9OT");
strcpy(bpmnames[20],"L:BPV201");
strcpy(bpmnames[21],"L:BPV202");

/* store names of all dipoles */
strcpy(dipnames[0],"L:HT3IN ");
strcpy(dipnames[1],"L:HT4IN ");
strcpy(dipnames[2],"L:HT5IN ");
strcpy(dipnames[3],"L:HT7IN ");
strcpy(dipnames[4],"L:HT7OUT");
strcpy(dipnames[5],"L:VT3IN ");
strcpy(dipnames[6],"L:VT4IN ");
strcpy(dipnames[7],"L:VT5IN ");
strcpy(dipnames[8],"L:VT7IN ");
strcpy(dipnames[9],"L:VT7OUT");
```

```
/* put a message on screen so people don't get in a hurry */
strcpy(tempstr,"Reading in Data - just a moment please");
btvm_c(nrows/2,ncols/2-strlen(tempstr)/2,tempstr,strlen(tempstr),YELLOW);

/* read in files from disk */
readcalib();
readzero();

/* erase message */
window_erase_line_c(WMNGR_BACKGROUND,nrows/2);

/* run through makelist once to initialize - get device indexes */
makelist();

} /* routine pgm_ini */

/*********************************************
/* pgm_kbd - Decode keyboard interrupts */

static void pgm_kbd(void)
{
    void    avgchange(void); /* change value of # pulses to avg */
    void    author(void);   /* put up nice display of author's name */
    void    calib(void);   /* do a calibration run */
    void    calcdipset(void); /* calculate new dipole settings */
    void    makeset(void); /* make the settings */
    void    expertlevel(void); /* go to expert level */

    if (binfld_c(row_list,col_list,strlen(liststr)) && expert == 1) {
        /* change active elements */
        makelist();
        return; }

    if (binfld_c(row_avg,col_avg,strlen(avgstr))) {
        /* reset number of readings to average */
        avgchange();
        return; }

    if (binfld_c(row_check,col_check,strlen(checkstr))) {
        /* read bpm's and calculate avg and std deviation */
        readbpm();
        return; }

    if (binfld_c(1,(ncols-strlen(titlestr))/2,strlen(titlestr))) {
        /* author message*/
        author();
        return; }

    if (binfld_c(row_calib,col_calib,strlen(calibstr))) {
        /* vary dipole correctors and watch bpm readings */
        calib();
        return; }

    if (binfld_c(row_calc,col_calc,strlen(calcstr))) {
        /* calculate new dipole settings */
        calcdipset();
        return; }

    if (binfld_c(row_calc+1,col_calc,strlen(makestr))&&makeflag==1) {
        /* make the new dipole settings */
        makeset();
        return; }
```

93/05/24
13:22:10

pal277.c

6

```
if (binfld_c(row_zero,col_zero,strlen(zerostr)) && expert == 1) {
    /* change zero values of BPM's i.e. location to steer to */
    makezero();
    return; }

if (binfld_c(row_cdat,col_cdat,strlen(cdatstr)) && expert == 1) {
    /* change calib data by hand */
    makecalib();
    return; }

if (binfld_c(1,ncols-strlen(novicestr)-2,strlen(novicestr))) {
    /* go to expert level or back to novice level */
    expertlevel();
    return; }

} /* routine pgm_kbd */

/*********************************************
/* pgm_per - Decode periodic interrupts */

static void pgm_per(void)
{ return; }

/*********************************************
/* pgm_trm - Program termination */

static void pgm_trm(void)
{ return; }

/*********************************************
static void author(void)
{ acknowledge_c(5,5,"Linac Steering\0Author: Kevin Junck\0           AD/Linac\0           x8368"
,65);
  return; }

/*********************************************
/* avgchange - change number to be averaged */

static void avgchange(void)
{ char tempavg[5],tempstr[6];

  makeflag=0;
  window_erase_to_eol_c(home_wid,row_calc+1,col_calc-1);

  /* read the device */
  inptxt_c(WMNGR_CENTER_IT,WMNGR_CENTER_IT,
            "# of Beam Pulses to Average",27,tempavg,4);

  numavg = atoi(tempavg);
  if(numavg > maxavg) {
    numavg = maxavg;
    sprintf(tempavg,"%d",numavg); }
  /* print on screen */
  btvm_c(row_avg,col_avg+strlen(avgstr),tempavg,strlen(tempavg),GREEN);
  /* kluge - sometimes crap gets displayed after average so print blanks */
  if (numavg < 100) {
    strcpy(tempstr,"      ");
    btvm_c(row_avg,col_avg+strlen(avgstr)+3,tempstr,strlen(tempstr),GREEN); }
  else {
    strcpy(tempstr,"      ");
    btvm_c(row_avg,col_avg+strlen(avgstr)+4,tempstr,strlen(tempstr),GREEN); }

} /* routine avgchange */
```

```
*****  
/* readbpm - read bpms and calculate average and standard deviation */  
  
static int readbpm(void)  
{  
    int      i,j,count,errcount,status,status2;  
    double   storage[maxbpm][maxavg+1];  
    double   sqrt(double);  
    static   void  do_15hz(void); /* request 15Hz studies beam */  
  
    if(makeflag==1) {  
        makeflag=0;  
        window_erase_to_eol_c(home_wid,row_calc+1,col_calc-1); }  
  
/* initialize */  
errcount = 1;  
count = 1;  
for (i=0; i<=numbpm-1; i++) {  
    avgbpm[i] = 0.0;  
    stddev[i] = 0.0; }  
  
/* output to screen */  
for (j=row_disp; j<= ROW_ERR-1; j++)  
    window_erase_line_c(WMNDR_BACKGROUND,j);  
for (j=0; j<=1; j++) {  
    for (i=0; i<= numbpm/2-1; i++)  
        if (bdindex[i+j*numbpm/2] > 0 ) {  
            /* print device name on Lexidata screen */  
            window_display_value_c(WMNDR_BACKGROUND,row_disp+i,  
                col_disp+j*(DEVNAM_LEN-3+8+2*len_disp+LEN_DEV_UNITS),  
                (void *)&bpm[(i+j*numbpm/2)*DEVNAM_LEN+2],CNV_CHAR,DEVNAM_LEN-2,CYAN);  
            window_display_value_c(WMNDR_BACKGROUND,row_disp+i,  
                col_disp+DEVNAM_LEN-2+  
                j*(7+2*len_disp+LEN_DEV_UNITS+DEVNAM_LEN-2),  
                (void *)" ",CNV_CHAR,3,CYAN); }  
/* also put up dipole settings */  
for (i=0; i<=numdipole-1; i++)  
    window_display_value_c(WMNDR_BACKGROUND,row_disp+i+1,  
        col_disp+2*(DEVNAM_LEN-2)+4*len_disp+16+2*LEN_DEV_UNITS,  
        (void *)&dipoles[i*DEVNAM_LEN+2],CNV_CHAR,DEVNAM_LEN-2,CYAN);  
window_display_value_c(WMNDR_BACKGROUND,row_disp,  
    col_disp+3*(DEVNAM_LEN-2)+4*len_disp+16+2*LEN_DEV_UNITS+2,  
    (void *)"Set",CNV_CHAR,3,CYAN);  
window_display_value_c(WMNDR_BACKGROUND,row_disp,  
    col_disp+3*(DEVNAM_LEN-2)+4*len_disp+16+2*LEN_DEV_UNITS+9,  
    (void *)"Read",CNV_CHAR,4,CYAN);  
status = -999;  
while (status != DIO_OK){  
    status = dio_get_lst(&dsetrlist,dipvalues,derrors,dunits); }  
for (i=0; i<=numdipole-1; i++)  
    window_display_value_c(WMNDR_BACKGROUND,row_disp+i+1,  
        col_disp+3*(DEVNAM_LEN-2)+4*len_disp+16+2*LEN_DEV_UNITS+1,  
        (void *)&dipvalues[i],CNV_FLOAT,5,GREEN);  
status = -999;  
while (status != DIO_OK){  
    status = dio_get_lst(&diplist,dipvalues,derrors,dunits); }  
for (i=0; i<=numdipole-1; i++)  
    window_display_value_c(WMNDR_BACKGROUND,row_disp+i+1,  
        col_disp+3*(DEVNAM_LEN-2)+4*len_disp+16+2*LEN_DEV_UNITS+8,  
        (void *)&dipvalues[i],CNV_FLOAT,5,GREEN);  
  
/* get bpm data */  
while (count <= numavg) {  
    if(errcount > beamwait) { /* timer timed out */
```

03/05/24
15:22:10

pal277.c

8

```
        error_message_c("No beam to be had",0,RED,TRUE);
        return 1;
    do_15hz();
    status = dio_get_lst(&bpmlist,bpmvalues,berrors,bunits);
    status2 = dio_get_dev_c(torind,piread,&tor,ftd);
    if (status == DIO_OK && status2 == DIO_OK) {
        window_display_value_c(WMNGR_BACKGROUND,row_check,
            col_check+strlen(checkstr)+2,
            (void *) &count,CNV_SHORT,3,YELLOW);
        if(fabs(tor) > 30.0) /* then there was beam */
            for (i=0; i<=numbpm-1; i++) {
                avgbpm[i]=avgbpm[i]+bpmvalues[i];
                storage[i][count]=bpmvalues[i]; }
            errcount=1;
            count++; } }
    else
        errcount++;
} /* while */

/* calculate average and standard deviation */
for (i=0; i<=numbpm-1; i++)
    avgbpm[i]=avgbpm[i]/numavg;

for (i=0; i<=numbpm-1; i++) {
    for (j=1; j<= numavg; j++)
        stddev[i]=stddev[i]+(storage[i][j]-avgbpm[i])*(storage[i][j]-avgbpm[i]);
    stddev[i] = stddev[i]/(numavg-1);
    stddev[i] = sqrt(stddev[i]); }

for (j=0; j<=1; j++) {
for (i=0; i<=numbpm/2-1; i++) {
    if(bdindex[i+j*numbpm/2] > 0) {
/* display the values on the background window */
    window_display_value_c(WMNGR_BACKGROUND,row_disp+i,
        col_disp+DEVNAM_LEN-2+3+j*(2*len_disp+7+LEN_DEV_UNITS+DEVNAM_LEN-2),
        (void *) &avgbpm[i+j*numbpm/2],CNV_DOUBLE,len_disp,GREEN);
    window_display_value_c(WMNGR_BACKGROUND,row_disp+i,
        col_disp+DEVNAM_LEN-2+3+len_disp+
        j*(2*len_disp+7+LEN_DEV_UNITS+DEVNAM_LEN-2),
        (void *) "+/-",CNV_CHAR,3,GREEN);
/* display the std dev on the background window */
    window_display_value_c(WMNGR_BACKGROUND,row_disp+i,
        col_disp+DEVNAM_LEN-2+len_disp+6+
        j*(2*len_disp+8+LEN_DEV_UNITS+DEVNAM_LEN-2),
        (void *) &stddev[i+j*numbpm/2],CNV_DOUBLE,len_disp,GREEN);
/* display the units on the background window */
    window_display_value_c(WMNGR_BACKGROUND,row_disp+i,
        col_disp+DEVNAM_LEN-2+7+2*len_disp
        +j*(LEN_DEV_UNITS+8+DEVNAM_LEN-2+2*len_disp),
        (void *) &bunits[(i+j*numbpm/2)*LEN_DEV_UNITS],CNV_CHAR,4,GREEN); } })
rbpmflag=1;
return 0; /* finished with no problem */
} /* routine readbpm */

*****  
/* calib - check bpm reading vs toroid current */

static void calib(void)
{ int statusd,statusset;
  int i,j,k,test,setcount,kj,klj;
  int analog = DIO_ANALOG;
  double dipole[numcalib+1];
  double bpmcalib[maxbpm][numcalib+1];
  double bpmstd[maxbpm][numcalib+1];
```

```
float origvalue[maxdipole];
float dipsetvalue[maxdipole];
double sumxx,sumy,sumx,sumxy,sumstd;
char tempstr[80],temp2str[10],abostr[24];
float initvalue,value,testvalue;
int numbppmpages,numdippages,bpmstart,bpmstop,dipstart,dipstop,doneflag;

makeflag=0;
window_erase_to_eol_c(home_wid,row_calc+1,col_calc-1);

/* put up panic button field */
sprintf(abostr,"%s"," Abort Calibration ");
btvm_c(row_calib+1,col_calib,abostr,strlen(abostr),RED);

/* initialize - get current settings */
statusd = dio_get_1st(&dsetrlist,origvalue,derrors);
if(statusd != DIO_OK){
    error_display_c("Cannot get current settings",ddindex[j],statusd);
    window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);
    error_message_c("Aborting Calibration",0,RED,TRUE);
    return; }
for (j=0; j<=numdipole-1; j++)
    dipsetvalue[j] = origvalue[j];

for (j=0; j<=numdipole-1; j++) /* loop through dipoles */

/* disable alarm scanning for dipole */
statusset = dio_disable(&ddindex[j],&analog);
if(statusset != DIO_OK)
    error_display_c("Problem disabling alarm ",ddindex[j],statusset);
else
    error_message_c("Everything is fine",0,CYAN,TRUE);

for (k=0; k<=numcalib; k++) /* loop through num of calib points */

/* check to see if panic button has been hit;
   if it has then reset all dipoles to original settings and return */
window_intype(&int_wid,&int_type,&int_row,&int_col,&int_info);
if (binfld_c(row_calib+1,col_calib,strlen(abostr))) {
    /* reset dipoles to current setting */
    for (klj=0; klj<=numdipole-1; klj++) {
        dipsetvalue[klj] = origvalue[klj];
        statusset = dio_set_dev_c(ddindex[klj],&dipsetvalue[klj]);
        if (statusset != DIO_OK)
            error_display_c("Problem resetting dipole ",ddindex[klj],statusset);
        statusset = dio_enable(&ddindex[klj],&analog);
        if (statusset != DIO_OK)
            error_display_c("Problem enabling alarm ",ddindex[klj],statusset);)}
    window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);
    error_message_c("Aborting Calibration",0,RED,TRUE);
    return; }

/* change dipole setting */
if (k == 0)
    dipsetvalue[j] = dipsetvalue[j] - (numcalib/2)*curstep;
else
    dipsetvalue[j] = dipsetvalue[j] + curstep;
/* make sure that dipset isn't too big */
if (dipsetvalue[j] > curlimit) dipsetvalue[j] = curlimit;
if (dipsetvalue[j] < -curlimit) dipsetvalue[j] = -curlimit;
dipole[k] = dipsetvalue[j];

/* print a warning on the screen */
strcpy(tempstr,"Changing dipole setting for ");
```

```
strncat(tempstr,&dipoles[j*DEVNAM_LEN],DEVNAM_LEN);
strcat(tempstr," to ");
sprintf(temp2str,"%f",dipsetvalue[j]);
strcat(tempstr,temp2str);
error_message_c(tempstr,0,YELLOW,TRUE);

/* DO SETTING */
statusset = dio_set_dev_c(ddindex[j],&dipsetvalue[j]);
if (statusset != DIO_OK) {
    error_display_c("Problem setting dipole",ddindex[j],statusset); }

/* check to see that new dipole setting has taken effect and stabilized.
Take 10 readings and make sure that they haven't changed by much. */
setcount = 0;
do {
    statusd = -999;
    while (statusd != DIO_OK){
        statusd = dio_get_lst(&diplist,dipvalues,derrors,dunits); }
    initvalue = dipvalues[j];
    for (kj=0; kj<=10; kj++) {
        statusd = -999;
        while (statusd != DIO_OK){ /* get another reading */
            statusd = dio_get_lst(&diplist,dipvalues,derrors,dunits); } }
    value = dipvalues[j];
    setcount++;
    if (setcount == 4) { /* try once more */
        statusset = dio_set_dev_c(ddindex[j],&dipsetvalue[j]);
        if (statusset != DIO_OK)
            error_display_c("Problem setting dipole",ddindex[j],statusset); }
    if (setcount == 8){ /* forget it */
        error_display_c("Giving up on setting dipole",ddindex[j],statusset);
        window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);
        error_message_c("Aborting Calibration",0,RED,TRUE);
        return; }
} while (fabs(value-initvalue) > 0.10);

/* go do bpm readings, if no beam then return to top level */
test = readbpm();
if (test == 1) {
    error_message_c("bpm reading error - aborting calibration",0,RED,TRUE);
    for (klj=0; klj<=numdipole-1; klj++) {
        dipsetvalue[klj] = origvalue[klj];
        statusset = dio_set_dev_c(ddindex[klj],&dipsetvalue[klj]);
        if (statusset != DIO_OK) {
            error_display_c("Problem resetting dipole ",ddindex[klj],statusset);}
        statusset = dio_enable(&ddindex[klj],&analog);
        if (statusset != DIO_OK) {
            error_display_c("Problem enabling alarm ",ddindex[klj],statusset);}
        window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);
        error_message_c("Aborting Calibration",0,RED,TRUE);
        return; }

/* store results */
for (i=0; i<=numbpm-1; i++) {
    bpmcalib[i][k] = avgbpm[i];
    bpmstd[i][k] = stddev[i]; }
} /* for k - done with one calibration point */

/* done taking calibration data for this dipole - reset to original value */
dipsetvalue[j] = origvalue[j];
/* SETTING */
statusset = dio_set_dev_c(ddindex[j],&dipsetvalue[j]);
if (statusset != DIO_OK) {
    error_display_c("Problem resetting dipole ",ddindex[j],statusset); }
```

93/05/24
1892510

pa1277.c

11

```
/* check to see that new dipole setting is ok */
setcount = 0;
do {
    statusd = -999;
    while (statusd != DIO_OK){
        statusd = dio_get_1st(&diplist,dipvalues,derrors,dunits); }
    initvalue = dipvalues[j];
    for (kj=0; kj<=5; kj++) {
        statusd = -999;
        while (statusd != DIO_OK){ /* get another reading */
            statusd = dio_get_1st(&diplist,dipvalues,derrors,dunits); } }
    value = dipvalues[j];
    setcount++;
    if (setcount == 4) { /* try once more */
        statusset = dio_set_dev_c(ddindex[j],&dipsetvalue[j]);
        if (statusset != DIO_OK)
            error_display_c("Problem setting dipole",ddindex[j],statusset); }
    if (setcount == 8){
        error_display_c("Giving up on setting dipole",ddindex[j],statusset);
        window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);
        error_message_c("Aborting Calibration",0,RED,TRUE);
        return; }
} while (fabs(value-initvalue) > 0.10);

/* successfully reset dipole to original setting - enable the alarm */
statusset = dio_enable(&ddindex[j],&analog);
if (statusset != DIO_OK) {
    error_display_c("Problem enabling alarm ",ddindex[j],statusset);
    window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);
    error_message_c("Aborting Calibration",0,RED,TRUE);
    return; }

/* calculate slope of line
y values are bpmcalib[i][0] bpmcalib[i][1]...
x values are dipole[0] dipole[1]... */
for (i=0; i<=numbpm-1; i++) {
    sumxx = 0.0;
    sumy = 0.0;
    sumx = 0.0;
    sumxy = 0.0;
    sumstd = 0.0;
    for (k=0; k<=numcalib; k++) {
        sumstd = sumstd + 1.0/bpmstd[i][k]/bpmstd[i][k];
        sumxx = sumxx + dipole[k]*dipole[k]/bpmstd[i][k]/bpmstd[i][k];
        sumx = sumx + dipole[k]/bpmstd[i][k]/bpmstd[i][k];
        sumy = sumy + bpmcalib[i][k]/bpmstd[i][k]/bpmstd[i][k];
        sumxy = sumxy + dipole[k]*bpmcalib[i][k]/bpmstd[i][k]/bpmstd[i][k];}
    slope[i+1][j+1] = (sumstd*sumxy-sumx*sumy)/(sumstd*sumxx-sumx*sumx);
    sigslope[i+1][j+1] = sqrt(sumstd/(sumstd*sumxx-sumx*sumx));
    if (fabs(slope[i+1][j+1]) < sigslope[i+1][j+1] ) slope[i+1][j+1] = 0.0;
    if (fabs(slope[i+1][j+1]) < 0.250) slope[i+1][j+1] = 0.0; }

}/* for j - whew! done with that dipole */

/* erase abort message */
window_erase_to_eol_c(home_wid,row_calib+1,col_calib-1);

/* all done! print results */
strcpy(tempstr,"MORE");
btvm_c(row_disp-1,col_disp,tempstr,strlen(tempstr),RED);
numbpmppages = (numbpm-1)/10+1;
numdippages = (numdipole-1)/7+1;
for (j=1; j<=numdippages; j++)
```

```
for (i=1; i<=numbpmpages; i++) {
    if (i==numbpmpages && j==numdippages) {
        strcpy(tempstr,"DONE");
        btvm_c(row_disp-1,col_disp,tempstr,strlen(tempstr),RED); }
    bpmstart=(i-1)*10+1;
    bpmstop=i*10;
    if (bpmstop > numbpmp) bpmstop=numbpmp;
    dipstart=(j-1)*7+1;
    dipstop=j*7;
    if (dipstop > numdipole) dipstop=numdipole;
    displaycalib(bpmstart,bpmstop,dipstart,dipstop);
    doneflag=0;
    while (doneflag==0) {
        window_intype(&int_wid,&int_type,&int_row,&int_col,&int_info);
        if (int_type == INTKBD && binfld_c(row_disp-1,col_disp,4))
            doneflag=1; } }
    window_erase_to_eol_c(home_wid,row_disp-1,col_disp);

/* check against historical data */
for (i=1; i<=numbpmp; i++)
    for (j=1; j<=numdipole; j++) {
        if (cdata[i][j] != 0.0 && slope[i][j] != 0.0)
            if( fabs((slope[i][j]-cdata[i][j])/cdata[i][j]) > 0.15)
                error_message_c("Calibration discrepancy, using newer data",0,RED,TRUE);
        cdata[i][j]=slope[i][j]; }
    calibflag=1; /* calibration completed successfully */
} /* routine calib */

*****  
/* do_15hz - request 15 Hz beam */

int call_num=0;
void do_15hz(void)

{ int status;

    if ( beam_enable_di ) {
        if ( (call_num++)%2 )
            status = dio_on(&beam_enable_di);
        else
            status = dio_off(&beam_enable_di);
        if ( status != DIO_OK )
            error_display_c("Beam enable dio problems", ERR_ACNET, status); }
} /* routine do_15hz */

*****  
/* calcdipset - calculate new dipole settings after calibration */

static void calcdipset(void)

{ int i,j,k,status;
    void transpose();
    void invmatrix();
    double t[maxbpm+1][maxbpm+1],ttrans[maxbpm+1][maxbpm+1];
    double m[maxbpm+1][maxbpm+1],minv[maxbpm+1][maxbpm+1];
    double newdipole[maxdipole+1],offset[maxbpm+1];
    char tempstr[30];
    int numbpmpcol,bpmstart,bpmstop;
    int numdipcol,dipstart,dipstop;

/* initialization and flag checking */
if (rbpmflag == 0) { /* then no bpm data have been taken to correct */
    acknowledge_c(5,5,"Read BPMs first!!!",17);
    return;} }
```

```
if (calibflag == 0) /* calibration hasn't been run - use defaults */
    acknowledge_c(5,5,"Using historical BPM-Dipole calibration data!",45);

/* reduce noise in calculation by setting limits on what is considered
   a steering problem */
for (i=1; i<= numbpm; i++) {
    offset[i]=avgbpm[i-1]-zdata[i-1];
    if (fabs(offset[i]) < stddev[i-1]/2.0) offset[i]=0.0;
    if (fabs(offset[i]) < 0.30) offset[i]=0.0; }

/* clear the screen */
for (j=row_disp; j<= ROW_ERR-1; j++)
    window_erase_line_c(WMNGR_BACKGROUND,j);

window_display_value_c(WMNGR_BACKGROUND,row_disp,
1, (void *) "OFFSET",CNV_CHAR,6,CYAN);
window_display_value_c(WMNGR_BACKGROUND,row_disp+1,
2, (void *) "(mm)",CNV_CHAR,4,CYAN);

/* put up offset values */
numbpmcol = (numbpm-1)/18 + 1;
for (j=1; j<=numbpmcol; j++) {
    bpmstart=(j-1)*18+1;
    bpmstop=bpmstart+17;
    if (bpmstop > numbpm) bpmstop=numbpm;
    for (i=bpmstart; i<=bpmstop; i++) {
        window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-bpmstart),
        8+(j-1)*12, (void *) &bpm[(i-1)*8+2],CNV_CHAR,6,CYAN);
        window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-bpmstart),
        15+(j-1)*12, (void *) &offset[i],CNV_DOUBLE,4,GREEN); } }

/* make a copy of calib data in t
   change indices from 0 - numdipole-1 to 1 - numdipole */
for (i=1; i<= numbpm; i++)
    for (j=1; j<= numdipole; j++)
        t[i][j] = cdata[i][j];

/* find the transpose of t */
transpose(t,numbpm,numdipole,ttrans);

/* multiply ttrans x t */
for (i=1; i<=numdipole; i++)
    for (j=1; j<=numdipole; j++)
        m[i][j]=0.0;

for (i=1; i<=numdipole; i++)
    for (j=1; j<=numdipole; j++)
        for (k=1; k<=numbpm; k++)
            m[i][j] = m[i][j] + ttrans[i][k]*t[k][j];

/* take the inverse of m */
invmatrix(m,numdipole,minv);

/* multiply -1 x minv x ttrans */
for (i=1; i<=numdipole; i++)
    for (j=1; j<=numbpm; j++)
        m[i][j]=0.0;

for (i=1; i<=numdipole; i++)
    for (j=1; j<=numbpm; j++)
        for (k=1; k<=numdipole; k++)
            m[i][j] = m[i][j] + minv[i][k]*ttrans[k][j];

for (i=1; i<=numdipole; i++)
```

```
    for (j=1; j<=numbpm; j++)
        m[i][j] = -1.00*m[i][j];

/* multiply m x avgbpm array to get change in dipole setting */
    for (i=1; i<=numdipole; i++)
        newdipole[i] = 0.0;
    for (i=1; i<=numdipole; i++)
        for (j=1; j<=numbpm; j++)
            newdipole[i] = newdipole[i]+m[i][j]*offset[j];

/* print out results */
    status = -999;
    while (status != DIO_OK){
        status = dio_get_1st(&dsetrlist,dipvalues,derrors,dunits); }

/* dipvalues is new setting */
    for (i=0; i<=numdipole-1; i++)
        if (fabs(newdipole[i+1]) > 0.10)
            dipvalues[i] = dipvalues[i]+newdipole[i+1];

    window_display_value_c(WMNGR_BACKGROUND,row_disp,
        8+(12*numbpmcol), (void *) "DIPOLE ",CNV_CHAR,7,CYAN);
    window_display_value_c(WMNGR_BACKGROUND,row_disp+1,
        8+(12*numbpmcol), (void *) "SETTINGS ",CNV_CHAR,9,CYAN);
    window_display_value_c(WMNGR_BACKGROUND,row_disp,
        8+(12*numbpmcol)+17, (void *) "CHANGE",CNV_CHAR,6,CYAN);
    window_display_value_c(WMNGR_BACKGROUND,row_disp,
        8+(12*numbpmcol)+24, (void *) "NEW SET",CNV_CHAR,7,CYAN);

/* display results on screen */
    numdipcol = (numdipole-1)/17+1;
    for (j=1; j<=numdipcol; j++) {
        dipstart = (j-1)*17+1;
        dipstop = dipstart+16;
        if (dipstop > numdipole) dipstop=numdipole;
        for (i=dipstart; i<=dipstop; i++) {
            window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-dipstart+1),
                8+(12*numbpmcol)+9, (void *) &dipoles[(i-1)*8+2],CNV_CHAR,6,CYAN);
            if(fabs(newdipole[i]) > 1.0)
                window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-dipstart+1),
                    8+(12*numbpmcol)+9+8, (void *) &newdipole[i],CNV_DOUBLE,5,RED);
            else if (fabs(newdipole[i]) < 0.1)
                window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-dipstart+1),
                    8+(12*numbpmcol)+9+8, (void *) &newdipole[i],CNV_DOUBLE,5,GREEN);
            else
                window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-dipstart+1),
                    8+(12*numbpmcol)+9+8, (void *) &newdipole[i],CNV_DOUBLE,5,YELLOW);
            window_display_value_c(WMNGR_BACKGROUND,row_disp+(i-dipstart+1),
                8+(12*numbpmcol)+9+8+7, (void *) &dipvalues[i-1],CNV_DOUBLE,5,GREEN);
        }
    }

    btvm_c(row_calc+1,col_calc,makestr,strlen(makestr),RED);
    makeflag=1;

} /* routine calcdipset */

*****  
/* makeset - make new dipole settings after calculation */

static void makeset(void)

{ int j,status;
  float nominal,tolerance;
  int analog = DIO_ANALOG;
```

```
*****
error_message_c("Must set nominal and re-enable alarm by hand.",0,RED,TRUE);
return;
*****
```

```
/* make sure new settings should be made */
if (decide_c(5,5,"Make new dipole settings?",25,30,DECIDE_CANCEL,YELLOW)){
    for (j=0; j<=numdipole-1; j++) {

        /* disable alarm */
        status = dio_disable(&ddindex[j],&analog);
        if(status != DIO_OK)
            error_display_c("Problem disabling alarm ",ddindex[j],status);

        /* get old nominal and tolerance */
        status=dio_alarm_limits(&ddindex[j],&nominal,&tolerance,DIO_NOMTOL);
        if(status != DIO_OK)
            error_display_c("Problem finding nom/tol ",ddindex[j],status);

        /* make setting */
        status = dio_set_dev_c(ddindex[j],&dipvalues[j]);
        if (status != DIO_OK)
            error_display_c("Problem setting dipole",ddindex[j],status);

        /* set new nominal and tolerance */
        status = dio_salarm_lim(&ddindex[j],&dipvalues[j],&tolerance);
        if (status != DIO_OK)
            error_display_c("Problem setting nom/tol",ddindex[j],status);

        /* enable alarm */
        status = dio_enable(&ddindex[j],&analog);
        if (status != DIO_OK)
            error_display_c("Problem enabling alarm ",ddindex[j],status);
    }
} /* routine makeset */

*****
/* makelist - construct arrays, get device indices */

static void makelist(void)

{
    int i,j,status,inflag,outflag,err;
    int icb,icd;
    void displaynames(void);

    makeflag=0;
    window_erase_to_eol_c(home_wid,row_calc+1,col_calc-1);

    if (firsttime != 0) {
    /* get info on what bpms and dipoles are going to be active */
    outflag=0;
    while (outflag == 0) {
        inflag=0;
        displaynames();
        while (inflag == 0) {
            window_intype(&int_wid,&int_type,&int_row,&int_col,&int_info);
            switch (int_type) {
                case INTKBD: /* keyboard interrupt */
                    for (i=0; i<=(maxbpm-1)/2; i++){
                        if (binfld_c(row_disp+i+3,col_disp,DEVNAM_LEN)) {
                            bpmflag[i]=bpmflag[i]+1;
                            if (bpmflag[i]==2) bpmflag[i]=0; })
                        for (i=(maxbpm-1)/2+1; i<=maxbpm-1; i++) {
```

```
if (binfld_c(row_disp+i-((maxbpm-1)/2+1)+3,
              col_disp+DEVNAM_LEN+moveover,DEVNAM_LEN)) {
    bpmflag[i]=bpmflag[i]+1;
    if (bpmflag[i]==2) bpmflag[i]=0; })
for (i=0; i<=maxdipole-1; i++) {
    if (binfld_c(row_disp+i+3,
                  col_disp+2*DEVNAM_LEN+2*moveover,DEVNAM_LEN)) {
        dipflag[i]=dipflag[i]+1;
        if (dipflag[i]==2) dipflag[i]=0; )
    if (binfld_c(row_disp+1,col_disp,4))
        outflag=1;
    inflag=1; }
    break;
default:
    break;
} /* switch */
} /* while - inflag */
} /* while - outflag */
} /* skip this first time through code */

strcpy(bpms,"");
numbpm=0;
for (i=0; i<=maxbpm-1; i++)
    if (bpmflag[i] == 1) {
        numbpm++;
        strcat(bpms,bpmnames[i]); }
strcpy(dipoles,"");
numdipole=0;
for (i=0; i<=maxdipole-1; i++)
    if (dipflag[i] == 1) {
        numdipole++;
        strcat(dipoles,dipnames[i]); }

/* get device index numbers */
status = dio_device_index(bpms, bdindex, &numbpm, berrors);
if (status != DIO_OK) { /* failed in retrieving device index */
    error_message_c("Error retrieving bpm device index",0,RED,TRUE);
    return; }
for (j=0; j<=1; j++) {
for (i=0; i<= numbpm/2-1; i++)
    if (bdindex[i+j*numbpm/2] <= 0 ) { /* invalid device */
        error_message_c("Invalid bpm device name",0,RED,TRUE); }}

status = dio_device_index(dipoles, ddindex, &numdipole, derrors);
if (status != DIO_OK) /* failed in retrieving device index */
    error_message_c("Error retrieving dipole device index",0,RED,TRUE);
for (i=0; i<= numdipole-1; i++)
    if (ddindex[i] <= 0 ) /* invalid device */
        error_message_c("Dipole device name is screwed",0,RED,TRUE);

/* build lists */
/* bpm reading list */
status = dio_bld_get(&bpmlist,&numbpm, bdindex, &piread, berrors, &ftd);
if (status != DIO_OK) {
    err = error_in_list_c(status,numbpm,berrors);
    error_display_c("BPM list failed",ERR_ACNET,err);
    return; }
/* dipole readings list */
status = dio_bld_get(&diplist,&numdipole, ddindex, &piread, derrors, &ftd);
if (status != DIO_OK) {
    err = error_in_list_c(status,numdipole,derrors);
    error_display_c("Dipole get list failed",ERR_ACNET,err);
    return; }
status = dio_bld_get(&dsetrlist,&numdipole, ddindex, &piset, derrors, &ftd);
```

```
if (status != DIO_OK) {
    err = error_in_list_c(status,numdipole,derrors);
    error_display_c("Dipole set list for read failed",ERR_ACNET,err);
    return;
}
/* set the dipoles list */
status = dio_bld_set(&dipsetlist,&numdipole,ddindex,&piset,derrors);
if (status != DIO_OK) {
    err = error_in_list_c(status,numdipole,derrors);
    error_display_c("Dipole set list failed",ERR_ACNET,err);
    return;
}

/* erase screen */
for (i=row_disp; i<= ROW_ERR-1; i++)
    window_erase_line_c(WMNGR_BACKGROUND,i);

/* set up calibration data */
for (i=0; i<=maxbpm; i++)
    for (j=0; j<=maxdipole; j++){
        cdata[i][j]=0.0;
        slope[i][j]=0.0;

icb=1;
for (i=0; i<=maxbpm-1; i++)
    if(bpmflag[i] == 1) {
        icd=1;
        for (j=0; j<=maxdipole-1; j++) {
            if( dipflag[j] == 1) {
                cdata[icb][icd]=hist[i][j];
                slope[icb][icd]=hist[i][j];
                icd++; }
            if (icd != numdipole+1)
                error_message_c("Problem In Dipole Calib Data Storage",0,RED,TRUE);
        icb++; }
    if (icb != numbpm+1)
        error_message_c("Problem In BPM Calib Data Storage",0,RED,TRUE);

icb=0;
for (i=0; i<=maxbpm-1; i++)
    if(bpmflag[i] == 1){
        zdata[icb] = bpmzero[i];
        icb++; }
    if (icb != numbpm)
        error_message_c("Problem In BPM Zero Data Storage",0,RED,TRUE);

calibflag = 0;
rbpmflag = 0;
firsttime = 1;

} /* routine makelist */

/*****************************************/
/* displaynames - display all bpm and dipole names for selection */

static void displaynames(void)

{
    int i;
    char tempstr[28];

    for (i=row_disp; i<= ROW_ERR-1; i++)
        window_erase_line_c(WMNGR_BACKGROUND,i);
    strcpy(tempstr,"Green = Active Element");
    btvm_c(row_disp,col_disp,tempstr,strlen(tempstr),CYAN);
    strcpy(tempstr,"DONE");
    btvm_c(row_disp+1,col_disp,tempstr,strlen(tempstr),RED);
```

```
for (i=0; i<=(maxbpm-1)/2; i++) {
    if (bpmflag[i] == 1)
        btvm_c(row_disp+i+3,col_disp,bpmnames[i],DEVNAM_LEN,GREEN);
    else
        btvm_c(row_disp+i+3,col_disp,bpmnames[i],DEVNAM_LEN,YELLOW);}

for (i=(maxbpm-1)/2+1; i<=maxbpm-1; i++) {
    if (bpmflag[i] == 1)
        btvm_c(row_disp+i-((maxbpm-1)/2+1)+3,col_disp+DEVNAM_LEN+moveover,
                bpmnames[i],DEVNAM_LEN,GREEN);
    else
        btvm_c(row_disp+i-((maxbpm-1)/2+1)+3,col_disp+DEVNAM_LEN+moveover,
                bpmnames[i],DEVNAM_LEN,YELLOW);}

for (i=0; i<=maxdipole-1; i++) {
    if (dipflag[i] == 1)
        btvm_c(row_disp+i+3,col_disp+2*DEVNAM_LEN+2*moveover,
                dipnames[i],DEVNAM_LEN,GREEN);
    else
        btvm_c(row_disp+i+3,col_disp+2*DEVNAM_LEN+2*moveover,
                dipnames[i],DEVNAM_LEN,YELLOW);}

} /* routine displaynames */

//*********************************************************************
/* makezero - change bpm zeros */
static void makezero()

{ void displayzero(void);
  void writezero(void);
  int outflag,inflag,i,j;
  char tempstr[9];
  float atof(char[]);

  outflag=0;
  while (outflag == 0) {
    inflag=0;
    displayzero();
    while (inflag == 0) {
      window_intype(&int_wid,&int_type,&int_row,&int_col,&int_info);
      switch (int_type) {
        case INTKBD: /* keyboard interrupt */
          for (i=0; i<=numbpm/2-1; i++){
            if (binfld_c(row_disp+i+1,col_disp,DEVNAM_LEN-2+1+len_disp)) {
              inptxt_c(WMNGR_CENTER_IT,WMNGR_CENTER_IT,
                        "Enter New Value For BPM",23,tempstr,6);
              zdata[i] = atof(tempstr); } }

          for (i=numbpm/2; i<=numbpm-1; i++) {
            if (binfld_c(row_disp+i-numbpm/2+1,
                         col_disp+DEVNAM_LEN-2+1+len_disp+moveover,
                         DEVNAM_LEN-2+1+len_disp)) {
              inptxt_c(WMNGR_CENTER_IT,WMNGR_CENTER_IT,
                        "Enter New Value For BPM",23,tempstr,6);
              zdata[i] = atof(tempstr); } }

        if (binfld_c(row_disp,col_disp,4)) outflag=1;
        if (binfld_c(row_disp,col_disp+DEVNAM_LEN-2+1+moveover,2)) {
          if (decide_c(5,5,"Save these zero points?",23,30,
                       DECIDE_CANCEL,YELLOW)){
            if(numbpm != maxbpm )
              acknowledge_c(5,5,"Must be using all bpms!",23);
            else
```

```
        writezero();}

    inflag=1;
    break;
default:
    break;
} /* switch */
} /* while - inflag */
} /* while - outflag */

for (j=row_disp; j<= ROW_ERR-1; j++)
    window_erase_line_c(WMNGR_BACKGROUND,j);

} /* routine makezero */

*****  
/* displayzero - print out current zero values on screen */

static void displayzero()
{ int i,j;
char tempstr[5];

for (j=row_disp; j<= ROW_ERR-1; j++)
    window_erase_line_c(WMNGR_BACKGROUND,j);

strcpy(tempstr,"DONE");
btvm_c(row_disp,col_disp,tempstr,strlen(tempstr),RED);
strcpy(tempstr,"SAVE");
btvm_c(row_disp,col_disp+DEVNAM_LEN-2+1+moveover
    ,tempstr,strlen(tempstr),RED);

for (j=0; j<=1; j++)
    for (i=0; i<= numbpm/2-1; i++) {
        window_display_value_c(WMNGR_BACKGROUND,row_disp+1+i,
            col_disp+j*(DEVNAM_LEN-2+1+moveover+len_disp),
            (void *)&bpm[(i+j*numbpm/2)*DEVNAM_LEN+2],CNV_CHAR,DEVNAM_LEN-2,GREEN);
        window_display_value_c(WMNGR_BACKGROUND,row_disp+1+i,
            col_disp+DEVNAM_LEN-2+1+j*(len_disp+moveover+1+DEVNAM_LEN-2),
            (void *)&zdata[i+j*numbpm/2],CNV_FLOAT,len_disp,GREEN); }

} /* routine displayzero */

*****  
/* writezero - save new bpmzero data to file */

static void writezero()
{ int status,i,j;
short vun; /* virtual unit number of data file */
static char file_name[25];
char tempstr[9];

strcpy(file_name,"LINAC_MISC:PA1277BPM.D ");
status = fm_open_c(&vun,file_name,DIRECT,FMNGR_WRITE_ACCESS,TRUE);
if (status != FSHARE_OK)
    error_display_c("File Opening error",ERR_ACNET,status);

for (i=0; i<=maxbpm-1; i++){
    sprintf(tempstr,"%3f",zdata[i]);
    status = fm_write_c(vun,tempstr,strlen(tempstr),i+1);
    if (status != FSHARE_OK)
        error_display_c("File Writing error",ERR_ACNET,status);
}

status = fm_close_c(vun);
```

```
if (status != FSHARE_OK)
    error_display_c("File Closing error",ERR_ACNET,status);

} /* routine writezero */

/*********************************************
/* makecalib - change calib data */
static void makecalib()

{ void writecalib();
int found,outflag,inflag,i,j;
char tempstr[10],messstr[30];
float atof(char[]);
int numbppmpages,numdippages,bpmstart,bpmstop,dipstart,dipstop,doneflag;
int bpmpage,dippage;

strcpy(tempstr,"MORE");
btvm_c(row_disp-1,col_disp,tempstr,strlen(tempstr),RED);
numbppmpages = (numbpm-1)/10+1;
numdippages = (numdipole-1)/7+1;

outflag=0;
bpmpage=1;
dippage=1;
while (outflag == 0) {
    inflag=0;
    if (bpmpage==numbppmpages && dippage==numdippages) {
        strcpy(tempstr,"DONE");
        btvm_c(row_disp-1,col_disp,tempstr,strlen(tempstr),RED);
        strcpy(tempstr,"SAVE");
        btvm_c(row_disp-1,col_disp+6,tempstr,strlen(tempstr),RED); }
    bpmstart=(bpmpage-1)*10+1;
    bpmstop=bpmpage*10;
    if (bpmstop > numbpm) bpmstop=numbpm;
    dipstart=(dippage-1)*7+1;
    dipstop=dippage*7;
    if (dipstop > numdipole) dipstop=numdipole;
    displaycalib(bpmstart,bpmstop,dipstart,dipstop);

    while (inflag == 0) {
        window_intype(&int_wid,&int_type,&int_row,&int_col,&int_info);
        switch (int_type) {
            case INTKBD: /* keyboard interrupt */

                for (i=bpmstart; i<=bpmstop; i++)
                    for (j=dipstart; j<=dipstop; j++)
                        if (binfld_c(row_disp+1+(j-dipstart)*2,10+(i-bpmstart)*7,len_disp)) {
                            inptxt_c(WMNGR_CENTER_IT,WMNGR_CENTER_IT,
                                    "Enter New Value",15,tempstr,6);
                            slope[i][j] = atof(tempstr); }

                if (binfld_c(row_disp-1,col_disp,4)) {
                    if (bpmpage==numbppmpages && dippage==numdippages)
                        outflag=1;
                    else {
                        bpmpage++;
                        if (bpmpage > numbppmpages) {
                            bpmpage = 1;
                            dippage++; } } }

                if (binfld_c(row_disp-1,col_disp+6,2))
                    if (bpmpage==numbppmpages && dippage==numdippages) {
                        if (decide_c(5,5,"Save this calib data to file?",29,30,
                            DECIDE_CANCEL,YELLOW)){
```

pa1277.c

```

if(numdipole == maxdipole && numbpm == maxbpm) {
    strcpy(messstr,"This will take a minute.");
    btvm_c(row_disp-1,col_disp+12,messstr,strlen(messstr),YELLOW);
    writecalib();
    window_erase_to_eol_c(home_wid,row_disp-1,col_disp+12); }
else
    acknowledge_c(5,5,"Must use all bpms & dipoles!",28); } }

inflag=1;
break;
default:
break;
} /* switch */
} /* while - inflag */
} /* while - outflag */

for (j=row_disp-1; j<= ROW_ERR-1; j++)
    window_erase_line_c(WMNGR_BACKGROUND,j);

} /* routine makecalib */

/*********************************************
/* displaycalib - print out current calib values on screen */

static void displaycalib(bpmstart,bpmstop,dipstart,dipstop)
int bpmstart,bpmstop,dipstart,dipstop;

{ int i,j;

for (j=row_disp; j<= ROW_ERR-1; j++)
    window_erase_line_c(WMNGR_BACKGROUND,j);

for (i=bpmstart; i<= bpmstop; i++)
    window_display_value_c(WMNGR_BACKGROUND,row_disp,10+(i-bpmstart)*7,
                           (void *) &bpms[(i-1)*DEVNAM_LEN+2],CNV_CHAR,DEVNAM_LEN-2,CYAN);

for (j=dipstart; j<=dipstop; j++){
    window_display_value_c(WMNGR_BACKGROUND,row_disp+1+(j-dipstart)*2,1,
                           (void *) &dipoles[(j-1)*DEVNAM_LEN+2],CNV_CHAR,DEVNAM_LEN-2,CYAN);
    for (i=bpmstart; i<=bpmstop; i++){
        window_display_value_c(WMNGR_BACKGROUND,row_disp+1+(j-dipstart)*2,
                               10+(i-bpmstart)*7, (void *) &slope[i][j],CNV_DOUBLE,5,GREEN); } }

} /* routine displaycalib */

/*********************************************
/* writecalib - save new bpm-dipole calib data to file */

static void writecalib()
{ int status,i,j,recno;
short vun; /* virtual unit number of data file */
static char file_name[25];
char tempstr[10];

strcpy(file_name,"LINAC_MISC:PA1277CAL.D ");
status = fm_open_c(&vun,file_name,DIRECT,FMNGR_WRITE_ACCESS,TRUE);
if (status != FSHARE_OK)
    error_display_c("File Opening error",ERR_ACNET,status);

recno=1;
for (i=1; i<=maxbpm; i++){
for (j=1; j<=maxdipole; j++) {
    sprintf(tempstr,"%3f",slope[i][j]);
    status = fm_write_c(vun,tempstr,strlen(tempstr),recno);
    recno++; }
}
}

```

```
if (status != FSHARE_OK)
    error_display_c("File Writing error",ERR_ACNET,status);
}

status = fm_close_c(vun);
if (status != FSHARE_OK)
    error_display_c("File Closing error",ERR_ACNET,status);

} /* routine writecalib */

//****************************************************************
static void readcalib()
{
    int status,i,j,recno;
    short vun; /* virtual unit number of data file */
    static char file_name[25];
    char tempstr[10];
    float atof(char[]);

    strcpy(file_name,"LINAC_MISC:PA1277CAL.D ");
/* data file contents: first index is bpm number, 2nd is dipole
   each number is 8 bytes long (1 char = 1 byte)
   hist[0][0] hist[0][1] hist[0][2]...hist[0][maxdipole-1] hist[1][0]
   hist[1][1]...hist[maxbpm-1][maxdipole-1] */

    status = fm_open_c(&vun,file_name,DIRECT,FMNGR_READ_ACCESS,TRUE);
    if (status != FSHARE_OK)
        error_display_c("File Opening error",ERR_ACNET,status);

    recno=1;
    for (i=0; i<=maxbpm-1; i++) {
        for (j=0; j<=maxdipole-1; j++) {
            status = fm_read_c(vun,tempstr,sizeof(tempstr),recno);
            recno++;
            if (status != FSHARE_OK) {
                error_display_c("File Reading error",ERR_ACNET,status);
                fm_close_c(vun);
                return;
            }
            hist[i][j] = atof(tempstr); } }

    status = fm_close_c(vun);
    if (status != FSHARE_OK)
        error_display_c("File Closing error",ERR_ACNET,status);

} /* routine readcalib */

//****************************************************************
static void readzero()
{
    int status,i;
    short vun; /* virtual unit number of data file */
    static char file_name[25];
    char tempstr[9];
    float atof(char[]);

    strcpy(file_name,"LINAC_MISC:PA1277BPM.D ");
    status = fm_open_c(&vun,file_name,DIRECT,FMNGR_READ_ACCESS,TRUE);
    if (status != FSHARE_OK){
        error_display_c("File Opening error",ERR_ACNET,status);
        return; }

    for (i=0; i<=maxbpm-1; i++) {
        status = fm_read_c(vun,tempstr,sizeof(tempstr),i+1);
        if (status != FSHARE_OK) {
            error_display_c("File Reading error",ERR_ACNET,status);
            fm_close_c(vun); }
```

```

        return;
    bpmzero[i] = atof(tempstr); }

status = fm_close_c(vun);
if (status != FSHARE_OK) {
    error_display_c("File Closing error",ERR_ACNET,status);
    return;
}

} /* routine readzero */

*****  

/* expertlevel - change from novice to expert or vice versa */

void expertlevel()
{
    if (expert == 0) {
        expert = 1;
        btvm_c(1,ncols-strlen(novicestr)-2,expertstr,strlen(novicestr),YELLOW);
        error_message_c("Okay, but be careful. ",0,YELLOW,TRUE);
        btvm_c(row_list,col_list,liststr,strlen(liststr),GREEN);
        btvm_c(row_zero,col_zero,zerostr,strlen(zerostr),GREEN);
        btvm_c(row_cdat,col_cdat,cdatstr,strlen(cdatstr),GREEN);
    }
    else {
        expert = 0;
        btvm_c(1,ncols-strlen(novicestr)-2,novicestr,strlen(novicestr),GREEN);
        window_erase_to_eol_c(home_wid,row_list,col_list);
        window_erase_to_eol_c(home_wid,row_cdat,col_cdat);
        btvm_c(row_check,col_check,checkstr,strlen(checkstr),GREEN);
    }
}

*****  

/* transpose - take the transpose of a matrix */

void transpose(m1,irow,icol,m2)
/* m1 is a matrix with irow rows and icol cols
   m2 is the transpose of m1 and thus has icol rows and irow columns */
int irow,icol;
double m1[maxbpm+1][maxbpm+1],m2[maxbpm+1][maxbpm+1];

{ int i,j;
    for (i=1; i<= irow; i++)
        for (j=1; j<= icol; j++)
            m2[j][i]=m1[i][j];
} /* routine transpose */

*****  

/* invmatrix - calculate inverse of a matrix */

void invmatrix(a,N,ans)
/* a is the original square matrix of size N
   ans is the inverse of a */
int N;
double a[maxbpm+1][maxbpm+1],ans[maxbpm+1][maxbpm+1];

{ int i,j,indx[maxbpm+5];
    double col[maxbpm+5];
    void ludcmp(),lubksb();

    ludcmp(a,N,indx);
    for(j=1; j<=N; j++) {
        for(i=1; i<=N; i++) col[i] = 0.0;
        col[j]=1.0;
}

```

```
lubksb(a,N,indx,col);
for(i=1;i<=N;i++) ans[i][j]=col[i]; }

} /* routine invmatrix */

/*****************************************/
/* lubksb - from Numerical Recipes */

void lubksb(a,n,indx,b)
double a[maxbpm+1][maxbpm+1],b[];
int n,indx[];
{
    int i,ii,ip,j;
    double sum;

    ii=0;
    for(i=1; i<=n; i++) {
        ip=indx[i];
        sum=b[ip];
        b[ip]=b[i];
        if (ii)
            for (j=ii; j<=i-1; j++) sum -= a[i][j]*b[j];
        else if (sum) ii=i;
        b[i]=sum; }
    for (i=n; i>=1; i--) {
        sum=b[i];
        for (j=i+1; j<=n; j++) sum -= a[i][j]*b[j];
        b[i]=sum/a[i][i]; }
} /* routine lubksb */

/*****************************************/
/* ludcmp - from Numerical Recipes */

void ludcmp(a,n,indx)
int n,indx[];
double a[maxbpm+1][maxbpm+1];
{
    int i,imax,j,k;
    double big,dum,sum,temp;
    double vv[maxbpm+1],d;

    d=1.0;
    for(i=1; i<=n;i++) {
        big=0.0;
        for (j=1; j<=n; j++)
            if ((temp=fabs(a[i][j])) > big) big=temp;
        if (big == 0.0) {
            error_message_c("Problem in calculation - Singular matrix.",0,RED,TRUE);
            return; }
        vv[i]=1.0/big;

        for (j=1; j<=n; j++) {
            for (i=1; i<j; i++) {
                sum=a[i][j];
                for (k=1; k<i; k++) sum -= a[i][k]*a[k][j];
                a[i][j]=sum; }
            big=0.0;
            for (i=j; i<=n; i++) {
                sum=a[i][j];
                for (k=1; k<j; k++)
                    sum -= a[i][k]*a[k][j];
                a[i][j]=sum;
                if ((dum=vv[i]*fabs(sum)) >= big) {
                    big=dum;
```

```
    imax=i; } }
if (j!=imax) {
  for (k=1; k<=n; k++) {
    dum=a[imax][k];
    a[imax][k]=a[j][k];
    a[j][k]=dum; }
  d = -1.0*d;
  vv[imax]=vv[j]; }
indx[j]=imax;
if (a[j][j] == 0.0) a[j][j]=1.0e-20;
if (j!=n) {
  dum=1.0/(a[j][j]);
  for (i=j+1; i<=n; i++) a[i][j] *= dum; }
} /* ludcmp */
/*********************************************
```